mikroKLAN 12

P.1877/84

12 1984

informatyka

Programowanie mikrokomputerów

PLZ

Pamięci na dyskach optycznych

Spis treści rocznika 1984

WYDAWNICTWO NOT SIGMA

informatyka

12 1984

Nr 12
Miesięcznik Rok XIX
Grudzień 1984

Organ Komitetu Informatyki MNSzWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

KOLEGIUM REDAKCYJNE:

Mgr inż. Zbigniew GLUZA, dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (zastępca redaktora naczelnego), prof. dr hab. Leon ŁUKASZEWICZ (redaktor naczelny), mgr inż. Andrzej J. PIOTROWSKI, dr inż. Janusz ZALEWSKI

STALE WSPÓŁPRACUJĄ:

Mgr inż. Witold ABRAMOWICZ (Szwajcaria), mgr Adam B. EMPACHER, mgr Katarzyna ISAAK, dr Jacek OWCZARCZYK, mgr Marek SOBCZYK, dr Andrzej SZAŁAS, dr Jakub TATARKIEWICZ, mgr inż. Teresa WILCZEK

PRZEWODNICZĄCY RADY PROGRAMOWEJ:

Prof. dr hab. Tadeusz PECHE

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". 2305. Obj. 4,0 ark. druk. Nakład 4250 egz. T-39.

INDEKS 36124

Cena egzemplarza zł 75,—
Prenumerata roczna zł 900,—

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA CZASOPISM I KSIĄŻEK TECHNICZNYCH
SIGMA
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

SPIS TREŚCI ROCZNIKA 1984

PLZ — język programowania mikroprocesorów (1)
Wiesław Wilczyński 1

Uniwersalne asemblery dla mikrokomputerów
Ryszard K. Kott, Krzysztof Szwed 4

Metody realizacji programów uruchomieniowych dla mikroprocesorów
Ryszard Rybus 8

Problemy programowania mikroprocesorów
Rafał Zieleniewski 19

mikroKLAN 11

— NS 32032 — mikroprocesor 32-bitowy
— Współpraca z napędem dysków elastycznych PLx45D (1)
— Mikrokomputer IMZ-80
— Programator pamięci EPROM 2716
— Schemat ZX81

Z KRAJU 21

— Rada ds. ZŚTO. Sekcja SARP

ZE ŚWIATA 23

— Pamięci na dyskach optycznych
— Najeźdźcy ze wschodu

TERMINOLOGIA III okł.

— Czy nasze poglądy są kompatybilne?

W NAJBLIŻSZYCH NUMERACH:

- Artur Krępski o PASCALU MT+
- Zbigniew Banasik o normie IEEE — arytmetyka zmiennoprzecinkowa
- Ryszard Kott o BASICU dla mikrokomputerów
- Waldemar Kapuścik o projektowaniu sieciowych baz danych
- Konrad Jabłoński o języku BCPL
- Wacław Iszkowski o mechanizmach komunikacji
- Marek Dziedzic, Krzysztof Perycz i Jerzy Wiliński o emulatorze E6RM-E
- Andrzej I. Litwiniuk o kompilatorze LOGLANU dla MERY 400
- Zbigniew Kierzkowski i Jacek Małuszyński o współbieżnej aktualizacji bazy danych
- Janusz Zalewski o tendencjach w dziedzinie komputerowych systemów sterowania w 1983 roku

WIESŁAW WILCZYŃSKI
Warszawa

P. 1877/84

PLZ – język programowania mikroprocesorów (1)

Do niedawna programiści zajmujący się oprogramowaniem mikrokomputerów mieli do wyboru dwa języki programowania: BASIC i język asemblera. Sytuacja powoli zmienia się i obecnie dostępne są języki wysokiego poziomu klasy PASCALA. Język PLZ jest do niego bardzo zbliżony, jeżeli chodzi o struktury danych i syntaktykę, choć w porównaniu z PASCALEM ma ograniczenia dotyczące operacji zmiennoprzecinkowych, niektórych operacji na rekordach oraz operacji wejścia-wyjścia.

Prace nad tym językiem zostały zapoczątkowane w roku 1977 przez firmę ZILOG. Zamierzeniem projektantów nie było stworzenie uniwersalnego języka wysokiego poziomu, ale języka programowania dla mikroprocesorowych systemów sterujących, w których obliczenia numeryczne nie stanowią o całości zadania. Ich celem było opracowanie języka niezbyt rozbudowanego, ale łatwo rozszerzalnego i umożliwiającego otrzymanie efektywnego kodu maszynowego dla mikroprocesorów Z80 i Z8000.

Niewątpliwą zaletą języka PLZ jest oddzielna kompilacja modułów, co umożliwia programiście kontrolę nad przepływem informacji pomiędzy modułami oraz przyspiesza proces kompilacji.

STRUKTURY DANYCH

Każdy obiekt w PLZ ma swój identyfikator (nazwę), który stanowi ciąg liter, cyfr i znaków podkreślenia „—", zaczynający się od litery. Nazwy mogą być dowolnej długości, ale w większości implementacji 31 pierwszych znaków jest znaczących (znane są implementacje, w których 127 pierwszych znaków jest znaczących). Separatorami (o całkowicie równoważnym znaczeniu) są: znaki spacji, powrotu karetki, zmiany wiersza, zmiany strony, przecinek, średnik, dwukropek i znaki tabulacji.

Każdy obiekt zadeklarowany w programie ma swój typ: prosty (ang. simple), tablicowy (ARRAY), rekordowy (RECORD) lub zdefiniowany przez użytkownika (TYPE). Do typów prostych zaliczamy:

SHORT_INTEGER — liczby całkowite z zakresu [—128, +127]
BYTE — Liczby całkowite z zakresu [0, +255]
INTEGER — liczby całkowite z zakresu [—32768, +32767]
WORD — liczby całkowite z zakresu 0, 65535
typ wskaźnikowy (adres obiektu).

Typy proste całkowitoliczbowe mogą być użyte do przechowywania kodów znaków alfanumerycznych. W zależności od typu obiektu można przechować jeden lub dwa znaki, np.:

```
CHAR_1 BYTE:='A' ! zmienna CHAR_1, przechowuje kod litery A!
CHAR_2 WORD:='AB' ! zmienna CHAR_2 przechowuje napis AB!
```

```
1.  CONSTANT
2.      TRUE := XFF
3.      FALSE := 0
4.      A_MINUS_10 := 'A'-10
5.      BUFFER_SIZE := 100
6.      DOUBLE_BUFFER_SIZE := BUFFER_SIZE*2
```

Przykład 1

W PLZ istnieje możliwość odwołania do stałych, których typ jest zgodny z typem każdej zmiennej całkowitej. Stałe tworzy się przez zadeklarowanie. W linii 1 przykładu 1 podano zapowiedź deklaracji stałych. W linii 2 stałej o nazwie TRUE nadano wartość szesnastkową (znak %), w linii 3 stałej o nazwie FALSE — wartość 0, a w linii 4 stałej A_MINUS_10 — wartość kodu litery A pomniejszoną o 10 [1]).

Jedną z możliwości budowania złożonych struktur danych jest tworzenie tablic. Tablica jest obiektem zbudowanym ze zmiennych tego samego typu, posiadających wspólną nazwę. Operacja dostępu do elementu tablicy następuje przez wskazanie jego indeksu, który może być dowolnym wyrażeniem typu całkowitoliczbowego, np.:

```
COUNT ARRAY [4 SHORT_INTEGER]
CHAR ARRAY [72 BYTE]
HELP SHORT_INTEGER
...
HELP:=COUNT [3]
HELP:=COUNT [I + K]
```

Tablica o wymiarze N ma wskaźniki z zakresem [0, N—1], tak więc ostatni element tablicy CHAR ma indeks 71. Język PLZ nie nakłada ograniczeń co do liczby wymiarów tablic, możliwe jest deklarowanie tablic dwu-, trój- i wielowymiarowych, np.:

```
SCORE ARRAY [5,7,4 INTEGER]
```

Powyżej zadeklarowano 140 elementów typu INTEGER o wspólnej nazwie SCORE.

W odróżnieniu od tablic, rekord jest strukturą pozwalającą grupować obiekty różnych typów. Każdy ze składników rekordu nazywa się polem rekordu. Przykładowo:

```
MISSING_PERSON RECORD [NAME ARRAY [25 BYTE]
BIRTHDATE RECORD [MONTH, DAY, YEAR BYTE],
SEX BYTE, HEIGHT,
WEIGHT BYTE]
```

— zadeklarowano rekord o nazwie MISSING_PERSON, pierwszym polu nazwanym NAME i przedstawiającym tablicę 25 elementów typu BYTE, drugim polu o nazwie BIRTHDATE stanowiącym rekord (o trzech polach MONTH, DAY i YEAR typu BYTE), trzecim polu o nazwie SEX typu BYTE oraz czwartym i piątym (HEIGHT i WEIGHT) typu BYTE. Odwołania do poszczególnych pól rekordu mogą być wykonane jak do każdego obiektu tego typu. Operacja ta nazwana jest selekcją i polega na podaniu nazwy rekordu, a następnie nazwy pola poprzedzonego kropką, np.:

```
MISSING_PERSON.SEX:=MEN
MISSING_PERSON.NAME[I]:= 'K'
```

Możliwe jest deklarowanie tablicy rekordów, co bardzo ułatwia operowanie dużym zbiorem rekordów tego samego rodzaju, np.:

```
MISSING_PERSON ARRAY [2000 RECORD [Deklaracja rekordu]]
```

Definiowanie nowych typów polega na odwołaniu do innych typów, np. wymienionych wcześniej. Definicja zaczyna się słowem TYPE, po którym następuje nazwa i opis typu, np.:

```
TYPE COLOR BYTE
     CHAR BYTE
     PART RECORD [NUMBER WORD,
                  LOCATION_CODE BYTE,
                  COST, PRICE_LIST WORD].
```

Powyżej zdefiniowano trzy nowe typy COLOR, CHAR i PART. Przykład deklaracji zmiennych tych typów wygląda następująco:

---

[1]) Numeracja wierszy nie jest elementem programu

cz. 9574

```
RED, ORANGE, YELLOW  COLOR
CURRENT_CHAR         CHAR
PICTURE              ARRAY [150,300 COLOR]
NEW_PART             PART
```

Typ wskaźnikowy

Niekiedy wygodniejsze jest użycie wskaźnika zmiennej (tzn. jej adresu lub deskryptora), a nie samej zmiennej. Do tego celu przewidziano w PLZ typy wskaźnikowe. W deklaracji zmiennej typu wskaźnikowego należy podać nazwę zmiennej, operator wskaźnikowy „^" oraz typ wskaźnika. Wartość zmiennej typu wskaźnikowego jest nadawana jej przy użyciu operatora #.

```
1.   CHAR     ¬BYTE
2.   BYTE_POINTER    ¬BYTE
3.   WORD_POINTER    ¬WORD
4.   SHORT_POINTER   ¬SHORT_INTEGER
5.   CURRENT_CHAR    BYTE
6.   CURRENT_CHAR := 'A'
7.   CHAR := #CURRENT_CHAR
8.   NEXT_CHAR := CHAR¬
9.   BYTE_POINTER := NIL
```

Przykład 2

W liniach 1, 2 oraz 3, 4 przykładu 2 zadeklarowano trzy zmienne o różnych typach wskaźnikowych: BYTE, WORD, SHORT_INTEGER. Zmienną o nazwie CURRENT_CHAR typu BYTE zadeklarowano w linii 5, zaś w linii 6 nadano tej zmiennej wartość kodu litery A. W kolejnej linii zmiennej wskaźnikowej BYTE_POINTER nadano wartość wskazującą (adres) CURRENT_CHAR (wskazuje literę A). Instrukcja przypisania w linii 8 nadaje zmiennej NEXT_CHAR wartość równą kodowi litery A, zaś w ostatniej linii przykładu zmiennej wskaźnikowej BYTE_POINTER o typie wskaźnikowym BYTE — wartość początkową NIL, jedynej stałej typu wskaźnikowego dostępnej w PLZ. Stała NIL jest zgodna z każdym typem wskaźnikowym i nie wskazuje żadnego obiektu w programie.

Zmiennej typu wskaźnikowego można nadać wartość początkową, równą adresowi pierwszego znaku wskazanego tekstu, np.:

```
CHAR=:#'DOWOLNY TEKST'
```

Typ zmiennej CHAR jest zgodny z typem każdej zmiennej wskaźnikowej dotyczącej obiektu o rozmiarze bajtu.

Deklarowanie zmiennych typu wskaźnikowego jest możliwe dla wszystkich zmiennych typu prostego, włączając zmienne typu wskaźnikowego, a więc możliwy jest typ wskaźnikowy zmiennych typu wskaźnikowego, np.:

```
BYTE_POINTER ^BYTE:=#ALA
POINTER_TO_BYTE POINTER^ ^BYTE:=#ALA
```

W pierwszej linii zadeklarowano zmienną wskaźnikową BYTE_POINTER do zmiennej typu BYTE, o wartości początkowej równej adresowi zmiennej ALA. W kolejnym wierszu zadeklarowano zmienną wskaźnikową do zmiennej typu wskaźnikowego, o wartości początkowej równej adresowi zmiennej wskaźnikowej typu BYTE posiadającej wartość równą adresowi zmiennej ALA.
Zmienne typu wskaźnikowego mogą występować jako elementy tablic lub rekordów, np.:

```
INT_PTR ARRAY [100^INTEGER]
MESSAGE RECORD [LENGH BYTE, CHAR_PTR ^BYTE]
```

Powyższa deklaracja powoduje utworzenie następujących obiektów (w nawiasach podano typ obiektu):

```
INT_PTR/99/          (^INTEGER)
INT_PTR/99/^         (INTEGER)
MESSAGE.CHAR_PTR     (^BYTE)
MESSAGE.CHAR_PTR ^   (BYTE)
```

Rozważmy przykład deklaracji

```
A ARRAY [10 BYTE]
A_PTR    ^ARRAY [10 BYTE]
```

Powyższa deklaracja, choć legalna, nie ma sensu — ponieważ dwa obiekty „ARRAY [10 BYTE]" są różnych typów i A_PTR nigdy nie będzie wskazywał tablicy A. Prawidłowy sposób rozwiązania tego problemu polega na zdefiniowaniu nowego typu (przykład 3). W linii 2 zadeklarowano nowy typ A_10_BYTE, zaś w linii 4 zmienną A tego typu. W ostatniej linii przykładu 3 zadeklarowano zmienną A_PTR typu wskaźnikowego o typie A_10_BYTE, i nadano jej wartość (adres) zmiennej A, a więc tablicy A_10_BYTE. Typy obiektów A, # A i A_PTR są następujące:

```
A  /A_10_BYTE/
#A  /A_10_BYTE/
A_PTR  /^A_10_BYTE/
```

```
1.  TYPE
2.      A_10_BYTE ARRAY [10 BYTE]
3.  INTERNAL
4.      A         A_10_BYTE
5.      A_PTR     ¬A_10_BYTE := #A
```

Przykład 3

Często zdarza się, że wygodniej jest przeszukiwać tablicę w sposób liniowy (tj. przy użyciu zmiennej typu wskaźnikowego wskazującej kolejno każdy element tablicy), niż bardziej kosztowną metodą indeksowania elementów. Można to zrobić przy użyciu następującej deklaracji:

```
A ARRAY [10 BYTE]
B_PTR ^BYTE:=# A [0]
```

Zmienna B_PTR może wskazywać każdy obiekt typu BYTE, a więc i element tablicy A. W drugiej linii nadano mu wartość początkową równą adresowi pierwszego elementu tablicy A. Na zmiennych typu wskaźnikowego można wykonać tylko dwie operacje (oprócz instrukcji przypisania), zwiększania lub zmniejszenia ich wartości o rozmiar obiektu, który jest wskazywany (INC i DEC). W powyższym przykładzie, gdzie wartość zmiennej B_PTR jest równa adresowi pierwszego elementu tablicy A (o indeksie 0) wykonanie operacji:

```
B_PTR:=INC B_PTR
```

powoduje, że wartość B_PTR staje się równa adresowi kolejnego elementu tablicy A. Wykonanie operacji:

```
B_PTR:=DEC B_PTR
```

powoduje ponowne nadanie zmiennej B_PTR wartości równej adresowi pierwszego elementu tablicy A.

```
1.  TYPE
2.      STRING  RECORD [LENGTH BYTE, CHARS ¬BYTE]
3.  INTERNAL
4.      NEWMESSAGE      STRING
5.      MESSAGE ARRAY [1000 STRING]
6.      MES_PTR ¬STRING
7.      MES_PTR := #NEWMESSAGE
8.      MES_PTR := #MESSAGE[0]
```

Przykład 1

W przypadku użycia zmiennej typu wskaźnikowego do określenia rekordu należy także utworzyć nowy typ. Rozważmy przykład 4. W linii 2 zdefiniowano nowy typ STRING, a w następnych liniach zadeklarowano zmienną NEWMESSAGE typu STRING i tablicę 1000 elementów typu STRING (czyli rekordów). Zmienną wskaźnikową do elementów typu STRING zadeklarowano w linii 6, a w linii 7 nadano jej wartość równą adresowi zmiennej NEWMESSAGE, zaś w kolejnej linii — wartość równą adresowi pierwszego elementu z tablicy MESSAGE.

```
1.  TYPE
2.      NODE_PTR        ¬NODE
3.      NODE    RECORD [FIELD_1 BYTE,
                        FIELD_2 INTEGER,
                        LINK NODE_PTR]
4.  INTERNAL
5.      DATA_BASE       ARRAY [1000 NODE]
```

Przykład 5

Typu wskaźnikowego można użyć do budowy struktur listowych, jak w przykładzie 5. W rezultacie powyższych definicji i deklaracji, pole LINK każdego z rekordów NODE może wskazywać następny rekord typu NODE. W podanym przykładzie widoczne jest jedyne dopuszczalne odstępstwo od generalnej reguły, że odwołanie do obiektu lub typu musi być poprzedzone jego definicją lub deklaracją. W linii 2 zdefiniowano zmienną wskaźnikową typu NODE, który jest zdefiniowany dopiero w linii 3.

Zgodność typów

Zgodność typów w języku PLZ polega na spełnieniu jednego z następujących warunków:

- dwie zmienne typu definiowanego przez użytkownika są zgodnego typu wtedy, gdy mają tę samą nazwę typu
- dwie zmienne typu prostego, ale nie wskaźnikowego (SHOR_INTEGER, BYTE, WORD, INTEGER), są zgodnego typu wtedy, gdy mają tę samą nazwę typu
- tablice lub rekordy mają typ zgodny wtedy, gdy są deklarowane w tej samej liście deklaracji
- dwie zmienne typu wskaźnikowego są zgodnego typu wtedy, gdy związane są z obiektami o zgodnych typach.

```
1.  TYPE
2.      BUFFER  ARRAY [128 BYTE];
3.      DATA    RECORD [DAY, MONTH, YEAR BYTE]
4.  INTERNAL
5.      A,B     BUFFER
6.      C,D     ARRAY [128 BYTE]
7.      E       BUFFER
8.      F       ARRAY [128 BYTE]
9.      G       DATA
10.     H       DATA
11.     I,J     RECORD [DAY, MONTH, YEAR BYTE]
12.     K       RECORD [DAY, MONYH, YEAR BYTE]
```

Przykład 6

Każde dwa obiekty, które nie spełniają jednego z wyżej wymienionych warunków nie są zgodnego typu. W przykładzie 6 następujące zbiory zawierają zmienne o zgodnych typach: (A,B,E), (C,D), (G,H), (I,J), (F), (K).

Deklarowanie i inicjowanie zmiennych

Ponieważ zmienne nie są stałą częścią programu, na programiście spoczywa obowiązek deklarowania zmiennych, zanim nastąpi odwołanie do nich. Deklarowanie polega na użyciu tzw. słów zasięgu (ang. scope word) i podaniu listy nazw. Słowami zasięgu nazwy są następujące słowa języka: EXTERNAL, GLOBAL, INTERNAL i LOCAL. Przy deklarowaniu zmiennych prostych, tablic i rekordów można nadać im wartość początkową przez użycie instrukcji przypisania, np.:

```
INTERNAL X INTEGER:=125
GLOBAL   COUNT, LIMIT BYTE:= [0, 255]
INTERNAL LOMASK WORD:=%00FF
INTERNAL C1, C2, X BYTE:= ['A', 'B']
```

W ostatnim wierszu, zmiennej X nie nadaje się wartości początkowej. Możliwe są też inne warianty nadawania wartości początkowej, np.:

```
INTERNAL A,B,C, INTEGER:= [1, ?, 3]
```

W tym przypadku, zmienna B nie ma nadanej wartości początkowej, a dla deklaracji:

```
INTERNAL A,B,C...X INTEGER:= [1,3,...]
```

zmienna A=1, zaś zmienne B, C do X otrzymują wartość 3.

Inicjowanie tablic wygląda podobnie, przy czym w przypadku tablic wielowymiarowych — wartości początkowe wymienione po symbolach przypisania są nadawane elementom tablicy według zmieniającego się indeksu wiersza. Przykładowo — po deklaracji: INTERNAL LETTER ARRAY [2,2 BYTE] := ['A', 'B', 'C', 'D']
kolejnym elementom tablicy zostaną nadane następujące wartości:

```
LETTER [0,0] = 'A' , LETTER [0,1] = 'B'
LETTER [1,0] = 'C' , LETTER [1,1] = 'D'
```

W poniższym przykładzie:

```
INTERNAL X ARRAY [8 BYTE] := [5,6,7]
```

tylko trzy pierwsze elementy mają nadane wartości A [0] =5, A [1] =6, A [2] =7; pozostałe elementy są nieokreślone.

Zapis:

```
INTERNAL A,B ARRAY [100 BYTE] := [[] , [0, ....]]
```

powoduje, że tablica A nie ma nadanych wartości początkowych, zaś tablica B ma wszystkie elementy równe 0.

W kolejnym przykładzie deklaracji połączonej z inicjowaniem:

```
INTERNAL A ARRAY [*WORD] :=[125, 348, 75]
```

rozmiar tablicy jest ustalany przez kompilator [*] na podstawie listy wartości początkowych i będzie równy 3 WORD.

Inicjowanie rekordów jest połączeniem inicjowania typów prostych i tablic, natomiast zmienne zadeklarowane jako LOCAL oraz zmienne o typie zdefiniowanym przez użytkownika nie mogą być inicjowane według przedstawionych zasad. Nadanie im wartości może odbyć się tylko przez instrukcję przypisania w ciele programu lub procedury.

PODSTAWOWE OPERACJE

W PLZ następujące symbole służą do oznaczenia działań arytmetycznych:

\+ dla dodawania
— dla odejmowania lub negacji
/ dla dzielenia
* dla mnożenia
MOD dla operacji modulo
ABS dla uzyskania wartości bezwzględnej.

Ponadto zdefiniowano następujące operacje logiczne: AND, OR, XOR, NOT. Ponieważ język PLZ zaprojektowano do celów sterowania, ma on następujące ograniczenia:

— standardowa operacja dzielenia daje zawsze wynik całkowity np.:
5/2 = 2 i (—5) /2 = —2

— nie jest kontrolowany nadmiar, jaki może powstać w trakcie wykonywania działań

— nie jest kontrolowane wykonanie nielegalnych operacji np. dzielenie przez zero lub X MOD O oraz przekroczenie zakresu indeksów tablicy

— nie ma żadnych funkcji standardowych.

```
TYPE
    B_PTR   ¬BYTE
    N_PTR   ¬NODE
    NODE    RECORD (F1 BYTE, LINK N_PTR)
INTERNAL
    W WORD; B BYTE
    I INTEGER
    BP B_PTR; RP N_PTR
    ...
    W := WORD(I)
    BP := B_PTR(RP)
    B := BYTE(WORD(BP)+WORD(RP))
    BP := B_PTR(WORD(BP)+WORD(ABS(I)))
    W := W OR WORD(B)
    W := WORD(RP¬.F1)
```

Przykład 7

Niekiedy zachodzi konieczność użycia — w wyrażeniu lub instrukcji przypisania — zmiennych, które są różnych typów. W niektórych językach konwersja typów jest dokonywana automatycznie, zaś w przypadku języka PLZ programista musi świadomie dokonać tej operacji. Zdefiniowano następujące jednoargumentowe operatory konwersji typów: BYTE, SHORT_INTEGER, WORD, INTEGER, definicja typu. Operacje konwersji zilustrowano w przykładzie 7. W PLZ nie jest dopuszczalna konwersja typów strukturalnych (tablicowych i rekordowych).

```
TYPE
    R_REC   RECORD (F1 BYTE, F2, F3 ¬R_REC)
INTERNAL
    A       ARRAY [100 R_REC)
    B       R_REC
    I       INTEGER := 0
    STORAGE_SIZE    WORD
    NUMELTS BYTE
    ...
    STORAGE_SIZE := SIZEOF(A)+SIZEOF(B)+SIZEOF(I)
    NUMELTS := SIZEOF(A)/SIZEOF(A[0])
```

Przykład 8

Wielkość obszaru wyrażonego w bajtach, zajmowanego przez zmienne typu prostego, zdefiniowanego przez użytkownika, lub strukturalnego — może być określona przy użyciu operatora SIZEOF. Wynik użycia operatora SIZEOF należy traktować jako stałą ze wszystkimi tego konsekwencjami (przykład 8).

Instrukcja przypisania w PLZ może być zapisana następująco:

```
zmienna:=zmienna + wyrażenie
zmienna + = wyrażenie
```

Podobnie, zapis
zmienna := zmienna — wyrażenie
jest równoważny zapisowi
zmienna — = wyrażenie.

Na przykład:
I+=1 jest równoważne I:=I+1
X—=1 jest równoważne Y:=Y—1

Język PLZ nie ma wbudowanych operacji wejścia-wyjścia. Projektanci języka zostawili pod tym względem pełną swobodę użytkownikom, dołączając jedynie do dostarczanego kompilatora bibliotekę procedur wejścia-wyjścia umożliwiających w miarę standardową obsługę typowych urządzeń, jak: dyski, drukarki czy monitor ekranowy.

RYSZARD K. KOTT
Instytut Informatyki
Politechnika Warszawska

KRZYSZTOF SZWED
Instytut Maszyn Matematycznych
Warszawa

Uniwersalne asemblery dla mikrokomputerów

Gwałtowny rozwój sprzętu mikrokomputerowego stworzył zapotrzebowanie na narzędzia ułatwiające tworzenie oprogramowania wspomagającego dla nowych typów mikroprocesorów. Każda firma mikrokomputerowa oferuje swoim klientom tzw. system wspomagania (ang. development system), w skład którego — nie licząc sprzętu — wchodzą najczęściej m.in. asembler i symulator oraz program uzdatniający (ang. debugger). Jednakże, taki system jest użyteczny tylko dla procesora danej firmy. Z tego powodu w wielu ośrodkach, wyposażonych w różnorodny sprzęt mikrokomputerowy, na ogół pochodzący od różnych producentów, rozwija się uniwersalne narzędzia programowe nie związane z konkretnym typem mikroprocesora. Dążenie do uniezależnienia oprogramowania wspomagającego od sprzętu jest wśród użytkowników tak silne, że ostatnio opublikowano nawet propozycję uniwersalnego egzekutora czasu rzeczywistego dla mikrokomputerów!

Obecnie najbardziej rozpowszechnione są uniwersalne symulatory, uniwersalne asemblery i uniwersalne języki mikroprogramowania (asemblery mikroprogramowania). W systemach tych wystarczy podać definicję mikroprocesora (lub jego języka), aby automatycznie uzyskać pożądany symulator lub asembler. Oczywiście, dotyczy to również mikrokomputerów segmentowych (ang. bit-slice microprocessors), dla których te narzędzia są szczególnie użyteczne. Wynika to stąd, że dla każdego nowego procesora, złożonego np. z układów serii AM 2900, konieczne jest uzyskanie nowego asemblera (nie można kupić gotowego). Za pomocą wymienionych narzędzi jest to szybkie i łatwe.

Biorąc pod uwagę cechy funkcjonalne uniwersalnych asemblerów można je podzielić na:

— systemy uniwersalne względem sprzętu
— systemy uniwersalne względem sprzętu i języka symbolicznego.

W przypadku asemblera uniwersalnego względem sprzętu użytkownik definiuje listę rozkazów danego mikroprocesora wraz z mnemonikami (ang. mnemonics). Pozostałe elementy składni języka symbolicznego są ustalone (identyczne dla wszystkich generowanych języków) lub wynikają jednoznacznie z listy rozkazów. Natomiast dla asemblera uniwersalnego względem sprzętu i języka podaje się definicję listy rozkazów procesora, a także definicję języka asemblera, w jakim chcemy programować.

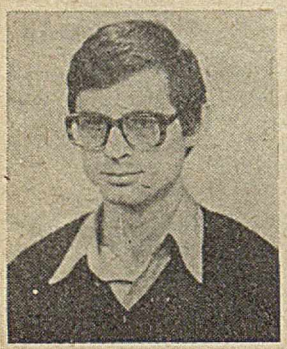
Z uwagi na sposób implementacji omawiane systemy można podzielić na:

— asemblery sterowane strukturą danych
— generatory asemblerów.

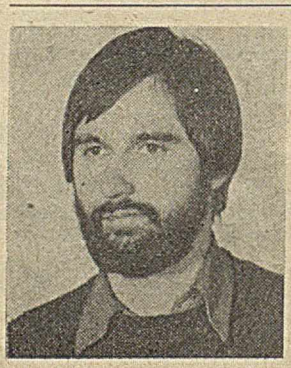
W porównaniu z asemblerami związanymi z konkretną maszyną, asemblery uniwersalne odznaczają się mniejszą efektywnością przetwarzania. Korzyść z zastosowania oprogramowania uniwersalnego polega na możliwości bardzo szybkiego uzyskania nowego asemblera (w ciągu kilku dni) przy niewielkim nakładzie pracy.

ASEMBLERY UNIWERSALNE WZGLĘDEM SPRZĘTU

Jeżeli nie ma oprogramowania uniwersalnego, to prostym sposobem uzyskania asemblera skrośnego na nowy mikroprocesor może być przerobienie innego asemblera skrośnego. O ile formaty instrukcji procesorów docelowych są takie same, przeróbka ogranicza się do wymiany tabeli mnemoników asemblera (mnemonik, jego kod i przyporządkowanie do grupy instrukcji). Taka jest właśnie zasada działania najprostszych uniwersalnych asemblerów skrośnych. Wszelako modyfikacja tabeli mnemoników jest w tym przypadku dokonywana automatycznie, na podstawie podanej definicji mikroprocesora.



Mgr inż. RYSZARD K. KOTT ukończył w 1976 r. Wydział Elektroniki Politechniki Warszawskiej (kierunek — informatyka). Obecnie pracuje w II PW jako starszy asystent. Specjalizuje się w zagadnieniach związanych z automatyczną produkcją oprogramowania dla systemów mikrokomputerowych.



Mgr inż. KRZYSZTOF SZWED ukończył studia w 1983 r. na Wydziale Elektroniki PW. Obecnie pracuje w Instytucie Maszyn Matematycznych w Warszawie.

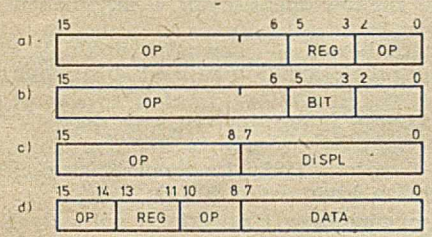
Koncepcja tych systemów opiera się na spostrzeżeniu, że formaty instrukcji poszczególnych mikroprocesorów 8-bitowych są na ogół podobne lub nawet identyczne. Najczęściej kod operacji zajmuje jeden bajt pamięci, po czym następuje 1- albo 2-bajtowy argument. Wystarczy więc wybrać odpowiednio reprezentatywny zbiór formatów instrukcji, aby można było definiować różne listy rozkazów.

Przykładem uniwersalnego asemblera skrośnego, działającego na tej zasadzie, jest system CASS8 [5]. Zdefiniowano w nim dziewięć klas instrukcji wskazywanych przez typ argumentu. Typ ten określa jednoznacznie długość instrukcji i sposób adresowania. W systemie istnieją procedury generowania kodu dla każdej klasy (instrukcje z tej samej klasy są przetwarzane jednakowo). Zdefiniowanie asemblera dla nowego mikroprocesora polega na opracowaniu tabeli mnemoników. Dla każdego mnemonika należy podać odpowiedni kod operacyjny i numer klasy, do jakiej należy instrukcja.

System taki, wskutek swej prostoty, ma istotne mankamenty. Wprowadzone uproszczenia (w CASS8 dopuszczono tylko jeden argument!) powodują, że informacja o trybie adresowania i (lub) używanym rejestrze musi być zakodowana w mnemoniku. W konsekwencji może to prowadzić do niewygodnej zmiany mnemoników instrukcji. Ponadto, przy założonej z góry liczbie typów instrukcji, klasa opisywalnych mikroprocesorów 8-bitowych zawsze będzie ograniczona. Na przykład — w CASS8 nie można opisać listy rozkazów mikroprocesora Z80, ponieważ zawiera ona instrukcje 4-bajtowe (z 2-bajtowymi kodami operacji).

Powyższe wady można usunąć, jeśli zezwolimy na definiowanie dowolnego formatu instrukcji mikroprocesora. Taki system może być naprawdę uniwersalny względem sprzętu, tzn. nie ma praktycznie żadnych ograniczeń na klasę definiowalnego sprzętu. Użytkownik najpierw definiuje nowy zbiór klas instrukcji, a następnie dopiero podaje definicje mnemoników wraz z ich kodami i numerami klas. Zazwyczaj nie występują klasy predefiniowane, toteż w celu dokonania definicji użytkownik sam musi podzielić listę rozkazów mikroprocesora na klasy. Za kryterium podziału należy przyjąć zgodność formatów instrukcji (p. rys. 1).

Asembler o takich możliwościach musi być bardziej złożony niż opisany poprzednio. W szczególności, w systemach sterowanych wczytaną strukturą danych musi istnieć możliwość przechowywania informacji o zdefiniowanych klasach instrukcji (np. w tablicy formatów).



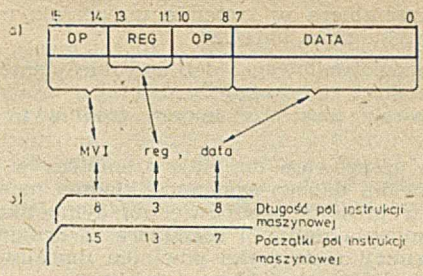
Rys. 1. Formaty wybranych instrukcji mikroprocesora Z80; Instrukcje a i b można zdefiniować w jednej klasie. Oznaczenia: OP — kod operacji, REG — kod rejestru, BIT — numer bitu, DISPL — przemieszczenie, DATA — argument bezpośredni

Przykładem generatora asemblerów dla dowolnego sprzętu jest system ASM/GEN [2]. Wykonanie definicji asemblera składa się w nim z następujących etapów:

— zdefiniowania adresowalnej jednostki pamięci (zazwyczaj jeden bajt)
— podziału listy rozkazów procesora na klasy
— zdefiniowania klas instrukcji
— określenia symbolicznych nazw rejestrów
— zdefiniowania (dla każdej klasy) mnemoników wraz z odpowiadającymi im kodami operacji.

Zdefiniowanie klasy polega na określeniu:

— długości instrukcji
— liczby pól (pole mnemonika i dowolna liczba argumentów)
— sposobu dzielenia argumentów dłuższych od jednostki adresowalnej (np. adresów 16-bitowych na bajty)
— przyporządkowania spójnych ciągów bitów instrukcji maszynowej do każdego pola instrukcji języka symbolicznego (rys. 2).



Rys. 2. Odwzorowanie pomiędzy polami instrukcji języka symbolicznego i bitami rozkazu maszynowego na przykładzie instrukcji MVI mikroprocesora 8080; a) struktura rozkazu maszynowego, b) odpowiedni fragment definicji asemblera. W ASM/GEN pole mnemonika definiowane jest zawsze jako pierwsze, po nim definiowane jest pole odpowiadające pierwszemu argumentowi instrukcji itd. Pola argumentów mogą nakładać się na pole mnemonika, ale nie na siebie. Przetwarzanie linii programu źródłowego od lewej strony do prawej zapewnia wygenerowanie właściwego kodu dla pola REG

Argumenty wyodrębnione z kodu źródłowego są numerowane od lewej strony do prawej (liczba argumentów może być dowolna), a definicja określa, z którym argumentem związane jest każde pole w danym formacie instrukcji.

Asembler, wygenerowany przez system ASM/GEN, składa się z części stałej oraz modułów, utworzonych na podstawie dostarczonej przez użytkownika definicji (jeden moduł dla każdej klasy instrukcji). Większość elementów składni języka symbolicznego (zbiór pseudooperacji, separatory pól instrukcji, składnia wyrażeń arytmetycznych, itp.) jest ustalona. Język źródłowy nie dopuszcza w argumentach znaków specjalnych określających tryb adresowania. Tak więc, podobnie jak w systemie CASS8, konieczne bywa rozszerzenie zbioru mnemoników instrukcji.

SYSTEMY UNIWERSALNE WZGLĘDEM SPRZĘTU I JĘZYKA SYMBOLICZNEGO

Wszystkie asemblery uzyskane za pomocą systemu o cechach opisanych w poprzednim punkcie mają podobne języki symboliczne. Czasem można to uważać za zaletę, np. gdy w danym ośrodku badawczym pragnie się umożliwić programowanie różnych mikroprocesorów w tym samym języku (istnieje nawet propozycja języka uniwersalnego dla mikroprocesorów [3], używanego już w ETH w Zurichu). Nie ma to również istotnego znaczenia w przypadku nowych konstrukcji, np. z mikroprocesorów segmentowych, gdyż wtedy potrzebny jest jakikolwiek asembler i jego postać jest mniej ważna.

Najczęściej jednak ustalony z góry język okazuje się niewygodnym ograniczeniem, ponieważ uniemożliwia korzystanie z istniejących podręczników czy też oprogramowania udostępnianego przez różne firmy. Ponadto programiści znający już dany mikroprocesor musieliby uczyć się nowego języka.

Zastosowanie asemblera uniwersalnego względem języka usuwa te kłopoty. Znane są rozwiązania takich systemów o rozmaitym stopniu ogólności, tj. o różnych ograniczeniach względem definiowanego języka symbolicznego. Asemblery z tej grupy są w pełni uniwersalne względem sprzętu (można definiować dowolne formaty rozkazów maszynowych), natomiast praktycznie w każdym z nich niektóre elementy składni języka symbolicznego są ustalone (wspólne dla wszystkich języków definiowalnych w danym systemie). W większości systemów przyjęto:

— ustalony zbiór (ale niekoniecznie niezmienną postać) pseudooperacji
— swobodny lub półswobodny (jedna instrukcja w linii) format tekstu
— stałą składnię wyrażeń arytmetycznych.

Niektóre elementy języka źródłowego zawsze można definiować.
Do nich należą:

— separatory pól instrukcji
— mnemoniki
— symboliczne nazwy rejestrów
— kodowanie trybów adresowania.

Pozostałe cechy języka, takie jak postać etykiety czy format stałych liczbowych i znakowych, są w poszczególnych rozwiązaniach definiowane lub ustalone.

Definicja asemblera w systemie uniwersalnym względem sprzętu i języka musi określać:

— format kodu źródłowego (definiowalne elementy)
— format kodu wynikowego
— odwzorowanie pomiędzy kodem źródłowym i wynikowym.

Użytkownik opracowuje definicję asemblera w metajęzyku specyficznym dla danego systemu. Składnia metajęzyka powinna, przez prostotę i odpowiednią symbolikę, zapewnić łatwość dokonania definicji asemblera. Z drugiej strony — zbiór konstrukcji metajęzyka powinien umożliwiać definiowanie jak najliczniejszej klasy języków.

W systemach opracowanych w latach sześćdziesiątych, format kodu źródłowego określano przez podanie zbioru produkcji gramatyki generującej dany język.

Podejście takie stosowano także do implementacji języków wysokiego poziomu w tzw. kompilatorach kompilatorów. Opis gramatyki, uwzględniający również akcje semantyczne translatora, sterował pracą systemu. We współczesnych rozwiązaniach uniwersalnych asemblerów niektóre typowe funkcje asemblera (np. operacje na tablicach symboli, rozbiór wyrażeń arytmetycznych, przetwarzanie pseudooperacji) są zdefiniowane wstępnie i ukryte przed użytkownikiem. Definiuje się jedynie akcje asemblera związane ze specyfiką danego języka symbolicznego oraz sprzętu.

Sformalizowana definicja asemblera jest przetwarzana przez system do postaci wewnętrznej struktury danych. System dostosowany do konkretnego mikroprocesora składa się z części stałej (programu — „szkieletu" translatora) oraz struktury danych zawierającej informacje o definiowanych cechach asemblera. Postać struktury może być różna, zależnie od implementacji, np. może to być zbiór tablic, struktura drzewiasta lub zbiór makrodefinicji. O ile nam wiadomo, nie zostało dotąd opublikowane rozwiązanie, zaimplementowane jako generator kodu.

Przykładem asemblera uniwersalnego względem sprzętu i języka symbolicznego jest opracowany przez nas system UAS (Uniwersalny Asembler Skrośny) [4].

DEFINIOWANIE ASEMBLERA W SYSTEMIE UAS

Języki symboliczne definiowalne w systemie UAS mają następujące wspólne cechy:

— ustalony zbiór pseudooperacji
— półswobodny format tekstu
— zbiór operatorów dopuszczalnych w wyrażeniach ograniczony do „+" i „—".

Pozostałe elementy składni asemblera koduje się w metajęzyku JDA (Język Definicji Asemblerów). Język ten powstał wskutek rozwinięcia koncepcji definiowania asemblera zarysowanej w publikacji [1]. Definicja składa się z sekcji określających cechy języka symbolicznego, sprzętu, a także ich wzajemne powiązania. Poniżej podano listę słów kluczowych, na podstawie których są identyfikowane poszczególne sekcje definicji:

$ALPHA — definicja dodatkowych liter
$RSX — definicja symboli zarezerwowanych asemblera
$SOURCE — definicja separatorów pól instrukcji (lub pseudoinstrukcji)
$LOC — definicja postaci odwołania do licznika adresów
$ADDR — definicja jednostki adresowej
$ZONE — definicja formatów stałych liczbowych i znakowych
$CHAR — definicja długości reprezentacji znaków i kodów znaków
$NOP — definicja kodu generowanego dla błędnych instrukcji
$ERR — definicja dodatkowych komunikatów asemblera wynikowego
$TYPE — definicja klasy instrukcji
$END — koniec definicji asemblera.

Sekcje występują w tekście definicji w kolejności zgodnej z uporządkowaniem tej listy. Możliwe jest pomijanie fragmentów definicji, a nawet całych sekcji. Zostaną wtedy domniemane elementy składni lub cechy sprzętu, np. formaty stałych zgodne z notacją języka CALM-2 i propozycją IEEE czy kod reprezentacji znaków ASCII. Może wystąpić dowolna liczba sekcji zawierających definicje klas instrukcji.

Większość nakładu pracy użytkownika, opracowującego definicję asemblera, przypada na podział zbioru instrukcji na klasy oraz określenie sposobu przetwarzania dla każdej z klas. Ze względu na proces przetwarzania, instrukcje zgrupowane w jednej klasie różnią się jedynie mnemonikami i zawartością pól stałych. Zatem warunkiem zgrupowania instrukcji w jednej klasie jest zgodność formatów ich postaci źródłowych i wynikowych oraz odwzorowań pomiędzy polami instrukcji symbolicznych i polami kodów wynikowych.

```
$TYPE   "SKOKI KROTKIE"
  %FLD  2, OPCODE(15,8), DISP(7,0)
  %MN   FLD(OPCODE), ('BCC',24), ('BCS',25), ('BEQ',27),
        ('BGE',2C), ('BGT',2E), ('BHI',22), ('BLE',2F),
        ('BLS',23), ('BLT',2D), ('BMI',2B), ('BNE',26),
        ('BVC',28), ('BVS',29), ('BPL',2A), ('BRA',20
        ('BSR',21)
  %OPD1 (U2)
        DISPL=OPD1-PC-2
```

Przykład 1

W przykładzie 1 podano definicję klasy w języku JDA — instrukcji skoków krótkich (względnych), dla mikroprocesora MC6800. Poszczególne części definicji sekcji mają następujące znaczenie:

%FLD — definiuje format instrukcji tej klasy jako dwubajtowy, złożony kolejno z pól OPCODE i DISPL

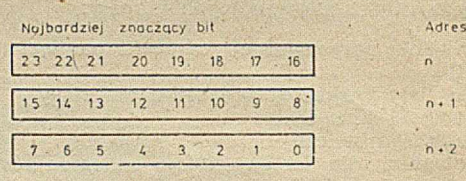
%MN — wskazuje, że mnemonikowi instrukcji źródłowej odpowiada w rozkazie pole OPCODE, a także podaje możliwe mnemoniki oraz ich kody

%OPD, wraz z następującym po nim podstawieniem, wskazuje, że argumentowi pierwszemu odpowiada pole DISPL, na które należy podstawić wartość podanego wyrażenia zakodowaną w kodzie — uzupełnienie dwójkowe.

W niektórych przypadkach użytkownik opracowujący definicję asemblera musi pamiętać o właściwym uporządkowaniu klas instrukcji. Przykładowo — instrukcje wirtualne, tj. generujące dla tego samego mnemonika kod o różnej długości, powinny być definiowane w dwóch klasach. Klasa zawierająca długie instrukcje wirtualne musi następować bezpośrednio za klasą grupującą ich krótkie odpowiedniki (oznaczoną przez opcję VIRT). Podobnie klasa zawierająca instrukcje o trybach adresowania określonych explicite (np. dla MC6800 przedrostek „#" określa adresowanie proste, przyrostek „(X)" — adresowanie indeksowe) musi poprzedzić klasę grupującą instrukcje o takich samych mnemonikach, ale bez znaków specjalnych w argumencie.

Ogólnie rzecz biorąc — wewnątrz definicji klasy mogą wystąpić konstrukcje metajęzyka JDA, o następującym znaczeniu:

— opcje klasy (np. informujące o tym, że instrukcje mają postać długą i krótką)
— definicja długości i podział na pola instrukcji maszynowych (za słowem kluczowym %FLD)
— definicja mnemoników i zawartości pól stałych instrukcji (zawierających kod operacji) rozpoczynająca się od słowa kluczowego %MN
— definicje argumentów (za słowem kluczowym %OPD) określające typ dopuszczalnych wyrażeń arytmetycznych oraz kod liczbowy
— definicje podstawień pod pola zmienne instrukcji
— definicje warunków (np. badanie tekstu źródłowego).



Rys. 3. Numeracja bitów instrukcji o długości trzech bajtów

Podział instrukcji maszynowej na pola wyodrębnia części zajmowane przez kod operacji, kody trybów adresowania poszczególnych argumentów i argumenty. Przyjętą konwencję numeracji bitów ilustruje rysunek 3. Definicja argumentu określa typ dopuszczalnych wyrażeń arytmetycznych (sumaryczny znak odwołań do etykiet i licznika adresów) oraz kod liczbowy (naturalny kod binarny, uzupełnienie dwójkowe, kod znak-moduł itp.), w którym zostanie przed-

stawiona obliczona wartość argumentu. Definicje podstawień pod pola zmienne i definicje warunków określają działanie asemblera po odszukaniu klasy, w której zdefiniowano mnemonik przetwarzanej instrukcji. Jeżeli argument instrukcji zawiera tylko wyrażenie arytmetyczne, należy zdefiniować podstawienie bezwarunkowe. W trakcie przetwarzania nastąpi w tym przypadku podstawienie obliczonej wartości argumentu pod wskazane pole (lub pola) instrukcji, np.:

F=OPD1

Asembler zinterpretuje pierwszy argument instrukcji jako wyrażenie arytmetyczne. Po obliczeniu wartości wyrażenia, o ile nie wystąpią błędy składniowe lub semantyczne, nastąpi konwersja do odpowiedniego kodu (określonego w definicji tego argumentu). Wynik zostanie wpisany do pola F (zdefiniowanego wcześniej dla tej klasy).

Jeżeli w polu argumentu mogą wystąpić znaki specjalne określające tryb adresowania lub jakiekolwiek symbole nie interpretowane jako składniki wyrażenia arytmetycznego, definiuje się podstawienie warunkowe, np.:

F=COND(SUBSTR(1, 1, OPD1) = '(' & SUBSTR(-1, -1-OPD1) = ')' ←SUBSTR(2,-2,OPD1), ←OPD1)

Jeżeli pierwszym znakiem argumentu jest "(", a ostatnim — ")", to tekst od drugiego do przedostatniego znaku tego argumentu zostanie zinterpretowany jako wyrażenie arytmetyczne. We wszystkich pozostałych przypadkach cały pierwszy argument będzie przetwarzany jak wyrażenie. Dalsze działanie asemblera jest identyczne jak w poprzednim przykładzie.

W przypadku, gdy chcemy zbadać postać argumentu (jednak bez generacji kodu binarnego), należy określić akcję asemblera za pomocą definicji warunku, np.:

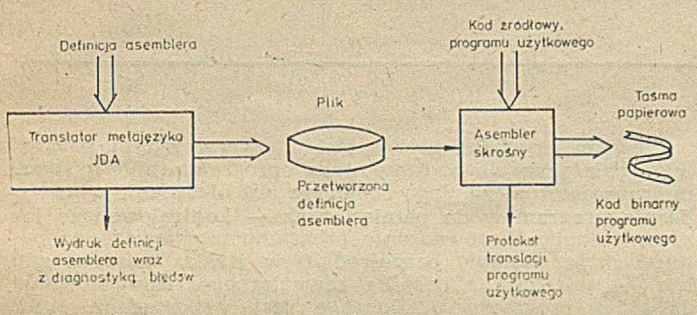
COND(OPD1='A')

Asembler zbada, czy pierwszy argument jest literą A (tj. np. nazwą symboliczną akumulatora) i będzie kontynuował przetwarzanie instrukcji w ramach danej klasy tylko wtedy, gdy warunek jest spełniony.

Inne przykłady konstrukcji JDA podano w Dodatku, zawierającym fragmenty definicji asemblera dla mikroprocesora MC6800 firmy MOTOROLA.

IMPLEMENTACJA SYSTEMU UAS

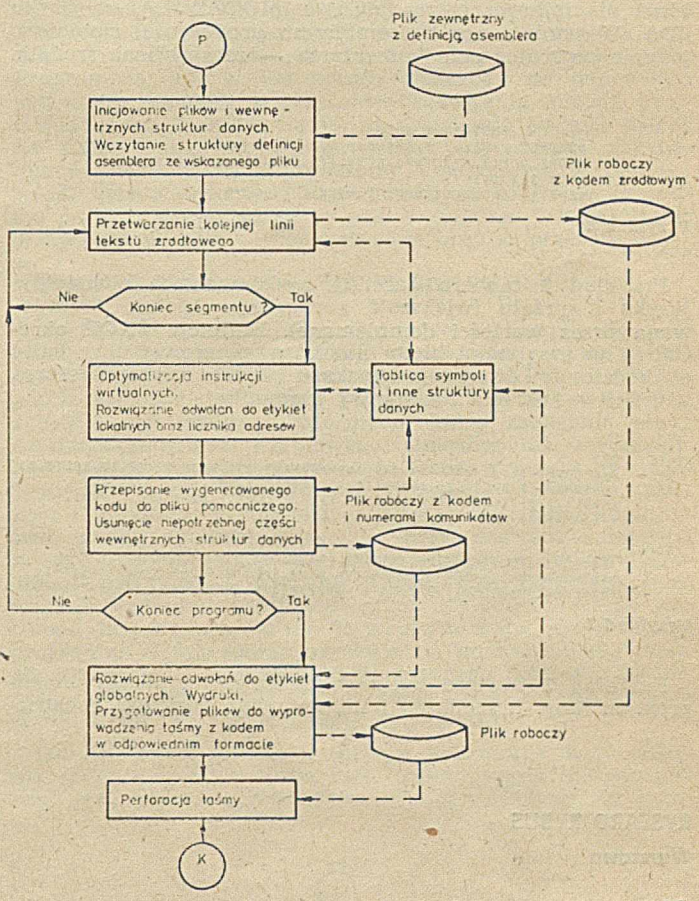
Wszystkie programy uniwersalnego asemblera napisano w języku PASCAL i uruchomiono na komputerze ODRA 1305 pod nadzorem systemu operacyjnego GEORGE-3 w Instytucie Informatyki Politechniki Warszawskiej. W ramach testowania systemu, uzyskano w krótkim czasie asemblery skrośne dla czterech typów mikroprocesorów: INTEL 8080, MC6800, PACE 6502, Z80.



Rys. 4. Wersja podstawowa systemu UAS

Podstawowa wersja systemu UAS składa się z translatora języka JDA i asemblera skrośnego (rys. 4). Translator metajęzyka przetwarza zakodowaną definicję asemblera, sprawdzając jej formalną poprawność, i składuje utworzoną strukturę danych na pliku. Asembler skrośny przetwarza program użytkowy do postaci nieprzesuwnego kodu binarnego w oparciu o strukturę definicji, odczytaną z pliku wskazanego przez użytkownika. Rozdzielenie etapów przetwarzania na fazę tłumaczenia programu użytkowego — umożliwia utworzenie biblioteki definicji asemblerów. Istnieje również wersja, w której obie części systemu zostały połączone w jeden program.

Definicja asemblera w postaci wewnątrzmaszynowej, jak również struktury danych tworzone w czasie translacji, składają się głównie z list i drzew. W celu uzyskania możliwie dużej szybkości przetwarzania, asembler skrośny został zrealizowany jako jednoprzebiegowy. Ponieważ dopuszcza się instrukcje wirtualne, zastosowano podwójny licznik adresów (minimalna wartość — dla krótkich wersji tych instrukcji, maksymalna — dla wersji długich). Wszystkie odwołania od etykiet i do licznika adresów są ostatecznie ustalane po optymalizacji instrukcji wirtualnych (po przetworzeniu segmentu kodu źródłowego).



Rys. 5. Schemat sterowania asemblera skrośnego w systemie UAS

Ogólny schemat sterowania asemblera skrośnego przedstawiono na rysunku 5. Dotychczasowe doświadczenia pozwalają już na charakterystykę ilościową systemu UAS. Podstawą są tu cztery asemblery uzyskane w tym systemie dla mikroprocesorów: 8080 (INTEL), 6800 (MOTOROLA), Z80 (ZILOG) i 6500 (PACE). W każdym przypadku dążono do uzyskania asemblerá możliwie zbliżonego do asemblera producenta. Poza ustaleniami przyjętymi a priori (dotyczącymi pseudooperacji i operatorów arytmetycznych), wprowadzono jedynie drobne zmiany w składni języków symbolicznych.

Translator metajęzyka JDA zajmuje 14 867 słów pamięci. Średnia szybkość przetwarzania definicji asemblerów wyniosła ok. 1380 linii/min. Niektóre parametry asemblerów zestawiono w tabeli 1. Zajętość pamięci asemblera wyniko-

Tabela 1. Parametry asemblerów wynikowych

Typ mikroprocesora	Długość tekstu definicji asemblera (w liniach)	Zajętość pamięci przetworzonej definicji (w słowach)	Średnia szybkość przetwarzania*) (w liniach/min)
8080	83	1181	530
6800	111	2242	605
6500	100	1521	565
280	359	6153	455

*) Nie uwzględniono czasu sprowadzenia do pamięci operacyjnej przetworzonej definicji i czasu wyprowadzania taśmy z kodem binarnym

wego jest równa sumie długości programu (14 563 słów) oraz obszaru zajmowanego przez definicję i wewnętrzne struktury danych (tablica symboli oraz kod wygenerowany dla aktualnie przetwarzanego segmentu znajdują się w obszarze pamięci przydzielanej dynamicznie). W przypadku wersji połączonej systemu (o długości 23 071 słów), struktury przetworzonej definicji asemblera zajmują o 15—30% więcej pamięci niż podano w tabeli.

DODATEK

W Dodatku zamieszczono trzy fragmenty definicji asemblera dla mikroprocesora MC6800 (MOTOROLA) w języku JDA. W porównaniu z asemblerem producenta, zmieniono jedynie separator pola komentarza — ze spacji na średnik.

```
                "MOTOROLA 6800"
=SOURCE LABEL=' ',OPRND=' ',CMNTL='*'
=ZONE DEC(),HEX('$'),OCT('@'),BIN('%'),CHAR('''')
=NOP 3,1,1,1
=ERR 1-'ARGUMENT MUSI ZAWIERAC NAZWE AKUM.: A, B'
```

Przykład 2

Przykład 2 zawiera definicje podstawowych elementów języka i sprzętu (większość parametrów zostaje zdefiniowana przez wartości domniemane). Definicja $NOP określa, że w przypadku błędu należy wygenerować trzy bajty o kodzie 1 (ilustracja rozkazu NOP mikroprocesora), a $ERR — określa dodatkowy komunikat.

```
=TYPE   "5"
 %FLD  2,OPCODE1(15,15),ACC(14,14),ADDRMODE(13,13),OPCODE2(12,8),ARG(7,0)
 %MN   FLD(OPCODE1,OPCODE2),('ADC',1,9),('ADD',1,0B),('CMP',1,1),
       ('SBC',1,2),('SUB',1,0),('AND',1,4),('EOR',1,8),('ORA',1,0A),
       ('BIT',1,5),('LDA',1,6)
 %OPD1
 %OPD2
       ACC=COND(OPD1='A'<0,OPD1='B'<1,ERR(1)
       ADDRMODE=COND(SUBSTR(1,1,OPD2)='#'<0,SUBSTR(-3,-1,OPD2)='(X)'<1)
       ARG=COND(SUBSTR(1,1,OPD2)='#'<SUBSTR(2,-1,OPD2),<SUBSTR(1,-4,OPD2))
```

Przykład 3

Przykład 3 zawiera definicje instrukcji z adresowaniem prostym (ang. immediate) i indeksowym. Ilustruje konstrukcje warunkowe JDA. W linii 43 bada się, czy argument zawiera nazwę symboliczną akumulatora A albo B. Jeśli nie, to drukowany jest komunikat. W liniach następnych bada się składnię adresowania prostego i indeksowego.

```
=TYPE   "11" (VIRT)
 %FLD  2,OPCODE1(15,15),ACC(14,14),OPCODE2(13,8),ARG(7,0)
 %MN   FLD(OPCODE1,OPCODE2),('ADC',1,19),('ADD',1,1B),('CMP',1,11),
       ('SBC',1,12),('SUB',1,10),('AND',1,14),('EOR',1,18),('ORA',1,1A),
       ('BIT',1,15),('LDA',1,16),('STA',1,17)
 %OPD1
 %OPD2
       ACC=COND(OPD1='A'<0,<1)
       ARG=OPD2
=TYPE   "12"
 %FLD  3,OPCODE1(23,23),ACC(22,22),OPCODE2(21,16),ARG(15,0)
 %MN   FLD(OPCODE1,OPCODE2),('ADC',1,39),('ADD',1,3B),('CMP',1,31),
       ('SBC',1,32),('SUB',1,30),('AND',1,34),('EOR',1,38),('ORA',1,3A),
       ('BIT',1,35),('LDA',1,36),('STA',1,37)%OPD1
 %OPD2
       ACC=COND(OPD1='A'<0,<1)
       ARG=OPD2
```

Przykład 4

Przykład 4 zawiera definicje dwóch klas opisujących przetwarzanie instrukcji wirtualnych mikroprocesora 6800. W klasie „11" zdefiniowano instrukcje krótkie odpowiadające danym mnemonikom, w klasie „12" — instrukcje długie odpowiadające tym samym mnemonikom.

LITERATURA

[1] Fujita S., Yoshida K.: A General Purpose Cross-Assembler for Microprocessors. Euromicro Newsletter, No. 3, 1977

[2] Johnson G., Mueller R.: ASM/GEN — A Cross-Assembler Generating System for Microcomputers. User's Manual Report ME76-1, Colorado State University, 1976

[3] Nicoud J. D.: CALM-2 — An Updated Common Assembly Language for 16- and 32-bit Microprocessors. Euromicro'80 Proceedings, North-Holland, 1980

[4] Szwed K.: Uniwersalny Asembler dla Mikrokomputerów. Praca dyplomowa, Instytut Informatyki, Politechnika Warszawska, 1983

[5] Tiberghien J., Taeymand J.: CASS8 — A Cross-Assembler Switchable for Most of the 8-bit Microprocessors. Euromicro Newsletter, No. 3, 1977.

RYSZARD RYBUS
Warszawa

Metody realizacji programów uruchomieniowych dla mikroprocesorów

Języki wysokiego poziomu, mimo niezaprzeczalnych zalet, nie są odpowiednie do wszystkich zastosowań. Realizacja operacji zależnych od maszyny, ograniczenia czasowe i pamięciowe mogą dyktować konieczność użycia języka asemblera. Jest to szczególnie widoczne przy programowaniu mikroprocesorów. W takim przypadku systemy mikroprocesorowe wyposaża się w programy uruchomieniowe (ang. debugger), które wspomagają programistę w procesie usuwania błędów z programu. Odpowiednia jakość programu uruchomieniowego ma duży wpływ na efektywność programowania, gdyż uruchamianie i testowanie oprogramowania zajmuje zwykle około 50% czasu przeznaczonego na przygotowanie programu.

Program uruchomieniowy jest konwersacyjnym programem należącym do podstawowego oprogramowania systemu mikroprocesorowego. Jego zadaniem jest ułatwienie wprowadzania programów do pamięci oraz — kontrolowanego ich uruchamiania. Wprowadzanie programu użytkowego do pamięci może nastąpić bezpośrednio w języku maszynowym lub — pośrednio z nośnika zewnętrznego. Kontrolowanie działania programu użytkowego jest wspomagane przez funkcje pozwalające na rozpoczęcie (wznowienie) i zatrzymanie programu. Realizacja tych funkcji wymaga zrealizowania mechanizmu, który umożliwia wymianę sterowania między programem użytkowym a programem uruchomieniowym.

Program uruchomieniowy pozwala programiście sprawdzać i modyfikować różne składniki stanu programu użytkowego, jak rejestry i wskaźniki procesora, wskazane miejsca pamięci itp. Kod programu uruchomieniowego najczęściej umieszcza się w programowanej pamięci stałej PROM. Obszary robocze tego programu znajdują się w pamięci RAM. Zbiór funkcji, którymi dysponuje program uruchomieniowy, określa wielkość zajmowanej przez niego pamięci. Proste programy mogą zajmować 256 bajtów, pamięci, bogatsze wersje zajmują 1—3 K bajtów.

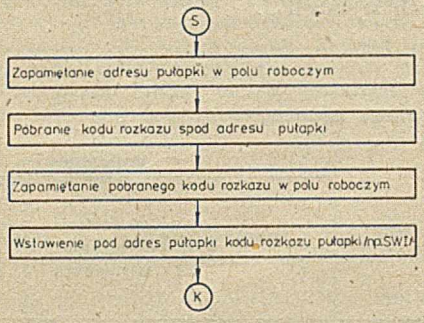
Życiorys i zdjęcie RYSZARDA RYBUSA zamieściliśmy w numerze 7/84

Poniżej przedstawiono dwie metody stanowiące podstawę do realizacji programów uruchomieniowych: mechanizm pułapek programowych oraz krokowe wykonywanie programu.

MECHANIZM PUŁAPEK PROGRAMOWYCH

W większości tradycyjnych programów uruchomieniowych stosowany jest mechanizm pułapek programowych [2]. Pułapka (ang. trap) jest to instrukcja, która powoduje przekazanie sterowania do programu uruchomieniowego (miejsce w programie użytkowym, w którym sterowanie jest przekazywane do programu uruchomieniowego nazywa się punktem wstrzymania; ang. breakpoint). Osiągnięcie pułapki powoduje zachowanie stanu programu użytkowego. Programista może wtedy dokonać sprawdzenia i ewentualnej modyfikacji stanu programu (tzn. rejestrów procesora i komórek roboczych) oraz błędnych instrukcji. Modyfikowanie stanu programu pozwala użytkownikowi wznowić wykonywanie programu od dowolnej instrukcji, z dowolną wartością stanu programu.

Istnieją dwa rodzaje pułapek. **Pułapki tymczasowe**, ustawiane przy uruchomieniu programu użytkowego, są aktywne do chwili ponownego przejęcia sterowania przez program uruchomieniowy. **Pułapki stałe** są ustawiane i likwidowane odpowiednimi dyrektywami. Do realizacji pułapek stosuje się 1-bajtowe rozkazy przerwań programowych. Przykładowo — mikroprocesor INTEL 8080 ma instrukcję RST, a MOTOROLA 6800 — instrukcję SWI.



Rys. 1. Algorytm ustawienia pułapki pod wskazanym adresem

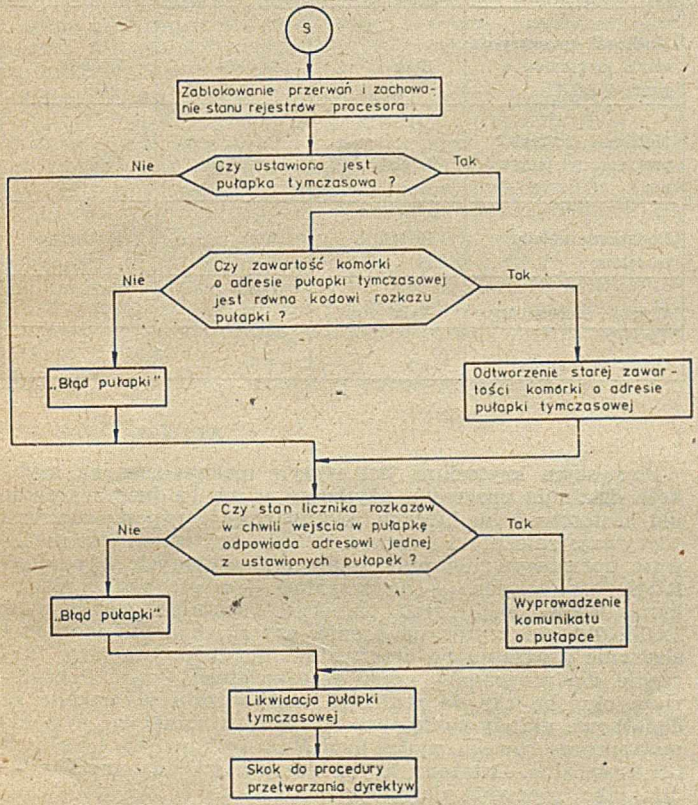
Na rysunku 1 przedstawiono schemat ilustrujący ustawienie pułapki. Szczegółowo czynności podejmowane przez program uruchomieniowy przy wejściu w pułapkę zilustrowano na rysunku 2.

Zaletą mechanizmu pułapek jest prostota realizacji. Nie wymaga on również żadnych dodatkowych układów sprzętowych. Istotne jest również, że program użytkowy jest wykonywany z pełną szybkością. Główną zaś wadą mechanizmu pułapek jest uzależnienie położenia pułapki od określonego przez użytkownika miejsca w programie. Jeśli użytkownik stwierdza błędne działanie programu po zatrzymaniu, to często nie może dokładnie określić źródła błędu. Obszar programu obejmujący miejsce powstania błędu jest zdefiniowany przez punkt ostatniego wznowienia programu i położenie pułapki, umożliwiającej wykrycie błędu. Programista jest w ten sposób zmuszony do ponownego ustawiania pułapek w celu zidentyfikowania miejsca powstania błędu. Z uwagi na to, że ustawienie pułapki polega na zmianie instrukcji w programie użytkowym, uruchamiane mogą być tylko programy umieszczone w pamięci RAM. Omawiany mechanizm umożliwia śledzenie programu po przerwaniu, ale tylko pod warunkiem wcześniejszego ustawienia pułapki w podprogramie obsługi przerwania.

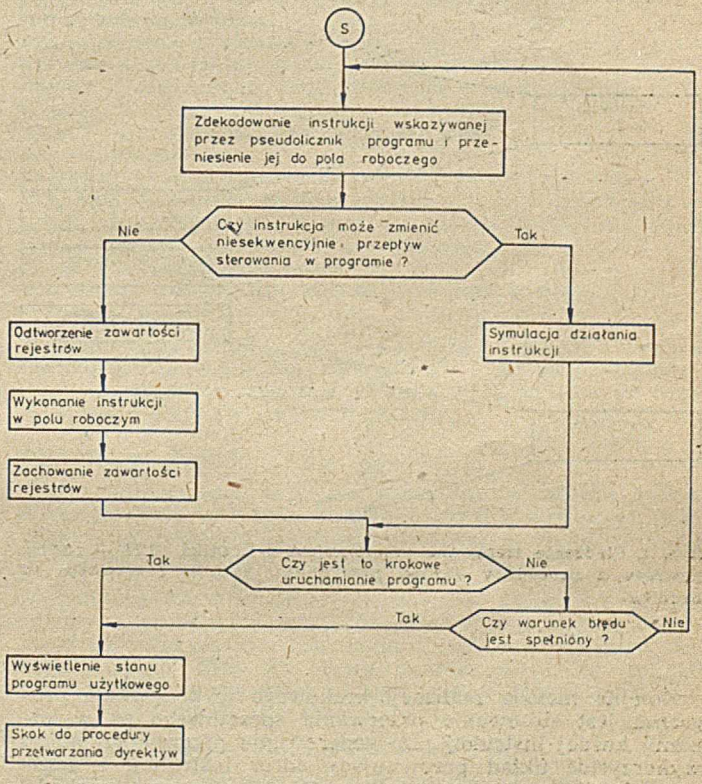
KROKOWE WYKONYWANIE PROGRAMU

Mechanizm umożliwiający krokowe wykonywanie programu (instrukcja po instrukcji) pozwala na realizację bardzo wydajnych narzędzi uruchomieniowych. Pełne wykonywanie krokowe jest często pożyteczne, ale dla większości sytuacji daje nadmiar informacji, który zaciemnia obraz działania programu użytkowego. Umożliwienie użytkownikowi uruchamiania warunkowego, które pozwala na określenie warunków wstrzymania programu, zwiększa szybkość i upraszcza uruchamianie. Użytkownik może zdefiniować błędną sytuację opisywaną przez logiczne zależności między różnymi składnikami stanu programu. Program uruchomieniowy po wykonaniu każdej instrukcji sprawdza wartość warunku określonego przez użytkownika. Jeśli warunek staje się prawdziwy, to wstrzymywane jest działanie programu. W ten sposób uzyskuje się uniezależnienie miejsca wtrzymania programu od konkretnego położenia instrukcji. Dzięki temu programista koncentruje się na naturze błędu, a nie na miejscu jego wystąpienia.

Istnieją różne metody realizacji krokowego wykonywania programu. Efekt krokowego wykonywania można uzyskać przy użyciu wcześniej omówionego mechanizmu puła-



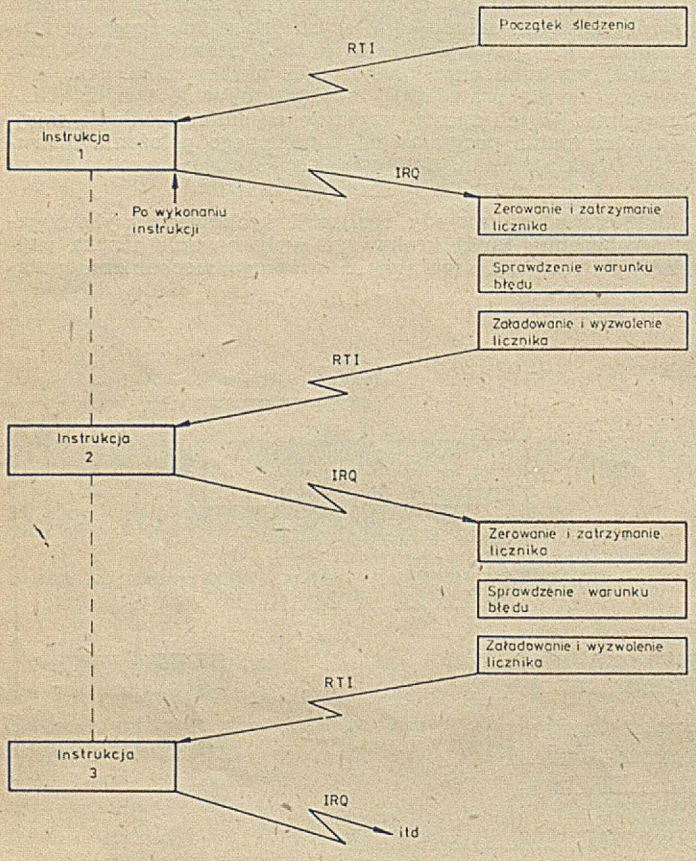
Rys. 2. Procedura obsługi wejścia w pułapkę (założono istnienie jednej pułapki tymczasowej)



Rys. 3. Schemat działania programu uruchomieniowego opartego na programowej realizacji krokowego wykonywania programu

pek programowych. Jednak w większości programów uruchomieniowych praca krokowa oparta o pułapki programowe nie została wykorzystana do realizacji uruchamiania warunkowego.

Metodą implementacji omawianego mechanizmu jest także programowa interpretacja programu użytkowego [1]. W metodzie tej kolejność wykonywania programu użytkowego jest określona przez wartość pseudolicznika programu. Sposób wykonania danej instrukcji przez procedurę krokowego wykonywania zależy od tego, czy efektem działania instrukcji może być niesekwencyjna zmiana przepływu sterowania w programie użytkowym. Instrukcje, które zachowują sekwencyjny porządek wykonywania programu, są wykonywane normalnie przez mikroprocesor. Natomiast działanie instrukcji, które mogą zmienić niesekwencyjnie przepływ sterowania w programie (np. instrukcje skoków, wywołania podprogramu) jest symulowane przez procedurę krokowego wykonywania. Polega to na odpowiedniej aktualizacji wartości pseudolicznika programu i innych składników stanu programu, które są zmieniane przez symulowaną instrukcję. Przykładowo — symulacja instrukcji wywołania podprogramu wymaga aktualizacji pseudolicznika programu, wskaźnika stosu oraz stosu. Schemat działania programu uruchomieniowego, opartego na programowej realizacji krokowego wykonywania, przedstawiono na rysunku 3. Oprócz wspomnianej możliwości uruchamiania warunkowego, opisana metoda pozwala na opracowywanie programów zapisanych w pamięci RAM i ROM, nie wymaga również żadnego dodatkowego sprzętu. Poważną jej wadą jest bardzo wolne wykonywanie programu użytkowego. Nie zapewnia śledzenia programu po przerwaniu.



Rys. 4. Śledzenie sterowane przerwaniem z licznika (RTI — rozkaz powrotu z procedury obsługi przerwania, IRQ — przerwanie od licznika)

Kolejną metodą realizacji krokowego wykonywania programu jest stosowanie przerwania sprzętowego po wykonaniu każdej instrukcji. Do generowania przerwania można wykorzystać układ porównujący adres instrukcji z adresem określonym przez użytkownika. Porównanie zachodzi dla ustalonych z góry bitów szyny adresowej, np. A_{11}—A_{15}. W efekcie każde pobranie instrukcji, której adres na bitach A_{11}—A_{15} jest zgodny z adresem zdefiniowanym przez użytkownika, powoduje wygenerowanie przerwania. Wadą tego rozwiązania jest konieczność zmiany adresu określonego przez użytkownika w zależności od położenia programu w pamięci. Inne rozwiązanie polega na zastosowaniu układu wykrywającego cykl pobrania pierwszego bajtu instrukcji (cykl FETCH). Stosowany jest również układ wykorzystujący prosty licznik sprzętowy (np. 74163) lub wygodniejszy programowalny czasomierz (np. MC 6840 firmy MOTOROLA lub 8253 firmy INTEL) [3]. Licznik jest używany do zliczania odpowiedniej liczby taktów zegara, tak aby ustawić sygnał przerwania przed zakończeniem wykonywania instrukcji. Przerwanie zostanie przyjęte przez mikroprocesor po wykonaniu instrukcji (rys. 4). Zaletą metody wykorzystującej przerwanie sprzętowe jest uproszczenie programu realizującego procedurę krokową. Możliwe jest uruchamianie programów zapisanych w pamięci ROM i RAM. Metoda ta zapewnia również śledzenie programu po przerwaniu. Jej wpływ na zwolnienie pracy programu użytkowego jest zdecydowanie mniejszy niż w przypadku metody interpretacji programu. Istnieje także możliwość uruchamiania programu z pełną szybkością. Zastosowanie przerwania maskowalnego pozwala na programowe blokowanie śledzenia programu. Programowalny czasomierz umożliwia wstrzymanie śledzenia na określoną liczbę taktów zegara. Główną wadą tej metody jest wprowadzenie dodatkowego sprzętu.

PORÓWNANIE PRZEDSTAWIONYCH MECHANIZMÓW

Do oceny przedstawionych mechanizmów realizacji programów uruchomieniowych zaproponowano pięć kryteriów wymienionych w tabeli 1.

Tabela 1. Porównanie mechanizmów realizacji programów uruchomieniowych

Kryterium oceny	Mechanizm		
	pułapki programowe	interpretacja programu użytkowego	sprzętowa realizacja krokowego wykonywania programu
Wpływ na szybkość działania programu	Mały ++	Duży	Średni +
Możliwość opracowywania programów w pamięci ROM	Brak	Istnieje ++	Istnieje ++
Możliwość śledzenia programu po przerwaniu	Istnieje +	Brak	Pełna ++
Dodatkowe układy sprzętowe	Brak ++	Brak ++	Konieczne
Wygoda uruchamiania	Mała	Duża ++	Duża ++
Ocena ogólna	+5	+6	+7

Pierwszym kryterium jest **wpływ mechanizmu na szybkość działania programu użytkowego;** ma to duże znaczenie dla zastosowań, w których istotną rolę w pracy programu odgrywają zależności czasowe. Drugim kryterium jest możliwość opracowywania **programów umieszczonych w pamięci ROM**, jest to ważne — na przykład — w przypadku mikrokomputerów, w których pamięcią programu jest pamięć ROM. Trzecie kryterium ocenia, **czy mechanizm dopuszcza śledzenie programu po przerwaniu**; ma to podstawowe znaczenie dla programów czasu rzeczywistego. Czwarte kryterium określa, **czy do realizacji mechanizmu wymagane są dodatkowe układy sprzętowe;** rozbudowa sprzętowa systemu mikroprocesorowego może być dla użytkownika uciążliwa i nieopłacalna. Ostatnim kryterium jest **wygoda uruchamiania**, jaką zapewnia dany mechanizm.

Dokończenie na str. 19

Proponowanie komukolwiek w naszym kraju konstrukcji bazującej na mikroprocesorze 32-bitowym może zostać poczytane za szaleństwo. Szczególnie gdy jeszcze nawet 16-bitowa technika znajduje się — oględnie mówiąc — w powijakach. A jednak proponuję przyjrzeć się bliżej najnowszej technice: prawda, że nie taka straszna! Sądzę, że zespół konstruktorów przymierzających się właśnie do 8086, zaopatrzony w odpowiednią literaturę (katalogi) i wcale nie tak znów bardziej wyrafinowane narzędzia niż te niezbędne w konstrukcjach 16-bitowych, doskonale by sobie poradził i z 32 bitami.

Skonstruowanie i produkcja 32-bitowego mikroprocesora zapewne jeszcze bardzo długo nie będzie możliwa w krajach RWPG. Natomiast wykorzystanie układów zachodnich o tej skali integracji wcale nie musi być odległe o dziesiątki lat.

Układ 32032 jest jednym z pierwszych przedstawicieli nowej generacji mikroprocesorów. Prawie wszyscy liczący się producenci układów mikroprocesorowych zapowiedzieli wprowadzenie jeszcze przed końcem tego roku własnych układów 32-bitowych. Czy znajdzie się w kraju grupa śmiałków, którzy zdołają przełamać dotychczasową niemoc?

NS 32032

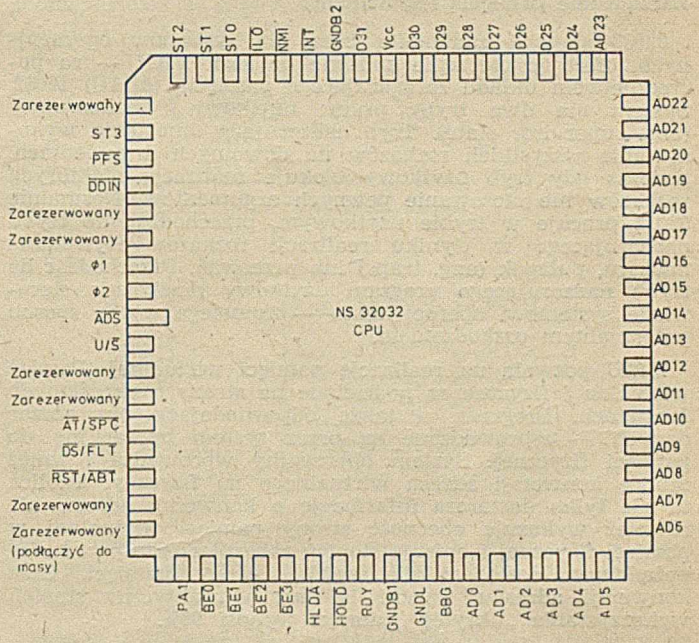
mikroprocesor 32-bitowy

Firma NATIONAL SEMICONDUCTOR reklamuje NS 32032 jako pierwszy dostępny na rynku rzeczywisty mikroprocesor 32-bitowy, posiadający zarówno 32-bitową zewnętrzną szynę danych, jak i 32-bitową architekturę wewnętrzną. Producent oferuje już wszystkie podzespoły potrzebne do utworzenia 32-bitowego systemu o możliwościach dużych maszyn cyfrowych. I to w cenie systemu mikroprocesorowego.

Bezsprzeczną zaletą „prawdziwych mikroprocesorów 32-bitowych jest większa ilość informacji przenoszenia w każdym cyklu odwołującym się do pamięci. Konsekwencją tego faktu jest większa dostępność magistrali systemowej dla innych procesorów, a więc lepsza efektywność przetwarzania współbieżnego w systemach wieloprocesorowych.

Firma NATIONAL SEMICONDUCTOR wprowadziła już wcześniej na rynek mikroprocesory 16032 i 08032, które posiadają wewnętrzną strukturę 32-bitową, ale ich komunikacja z systemem odbywa się poprzez odpowiednio: 16 i 8 bitową szynę danych. Mikroprocesory te są logicznie kompatybilne z 32032.

Opis wyprowadzeń 32032 przedstawia rysunek 1.



Rys. 1. Opis wyprowadzeń dla 32032
AD0—AD23 — szyna adresowa i danych
D24—D31 — pozostała część szyny danych
ADS — potwierdzenie danych
DDIN — sterowanie transmisją danych
BE0—BE3 — potwierdzenie z podobszarów pamięci
U/S — działanie w trybie nadzorującym/użytkowym
ILO — blokowanie szyn
PFS — początek wykonania rozkazu
ST0—ST3 — kod funkcji (status mikroprocesora)
AT/SPC — interpretacja adresu/sterowanie dwukierunkowego przerzutnika
DS/FLT — sterowanie: dane/stan dużej impedancji
PA1 — adres fizyczny
RST/ABT — zerowanie/przerwanie wykonywania rozkazu
INT, NMI — żądania przerwań: maskowalne i niemaskowalne
HOLD, HLDA — żądanie i przyjęcie żądania dostępu do szyny
RDY — synchronizacja układów peryferyjnych
Φ1, Φ2 — wyprowadzenia zegara
Vcc, GNDL, BBG, GNDB1, GNDB2 — zasilanie i masy

32-bitowa struktura

Zalety NS 32032 nie ograniczają się jedynie do szerokości szyny danych. W istocie mamy do czynienia z przedłużeniem logicznym struktury 32-bitowej na całą rodzinę NS 16000.

Poza zewnętrzną szyną danych, pozostałe bloki funkcjonalne 32032 są również 32-bitowe (rejestry, jednostka arytmetyczno-logiczna (ALU), wewnętrzne szyny danych i in), w przeciwieństwie do jednostek centralnych innych firm, w których jedynie niektóre z bloków funkcjonalnych są 32-bitowe, natomiast pozostałe pracują na 16 bitach. Dla wszystkich rozkazów argumenty mogą mieć format bajtu, słowa lub podwójnego słowa. Inaczej mówiąc — architektura jest w pełni symetryczna. Umożliwia ona również operacje zmiennoprzecinkowe, a przy zastosowaniu koprocesora arytmetycznego 16081 wszystkie rozkazy zmiennoprzecinkowe dopuszczają argumenty o pojedynczej lub podwójnej precyzji.

Architektura 32032 pozwala na dużą swobodę wyboru trybów adresowania dla całkowitych i zmiennoprzecinkowych argumentów rozkazów, co daje wyraźne korzyści przy stosowaniu języków wyższego poziomu.

Operacje na danych

Przy adresowaniu w trybie pośrednim w adresie można wydzielić dwa składniki: adres bazowy (zwykle w rejestrze) i przesunięcie (zwykle wchodzące w skład rozkazu) w stosunku do adresu bazowego.

Struktura 32032 umożliwia stosowanie przesunięć o bajty, słowa pojedyncze lub podwójne. W ten sposób otrzymuje się większą zwartość kodu i szybkość realizacji danego zadania. Przy 32 bitach rozmiar struktury danych jest ograniczony nie przez przesunięcie, a jedynie przez obszar adresowy jednostki centralnej.

Wykonanie działań zmiennoprzecinkowych

Działania w arytmetyce zmiennoprzecinkowej nabierają coraz większego znaczenia. Dlatego też do tego celu wykorzystywany jest koprocesor arytmetyczny 16081. Rozka-

zy i argumenty działań przejmuje on bezpośrednio z jednostki centralnej. Może wykonywać cztery podstawowe działania arytmetyczne, konwersje między danymi całkowitymi i zmiennoprzecinkowymi oraz porównania argumentów zmiennoprzecinkowych.

Zarządzanie pamięcią i systemem

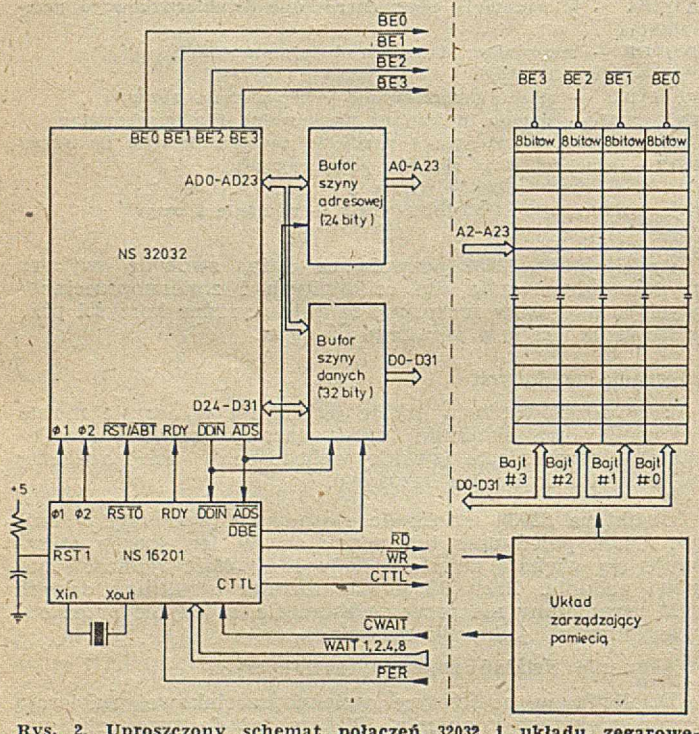
32032 stanowi znakomitą bazę dla systemów operacyjnych, oferując otoczenie zbliżone do serii VAX — za pośrednictwem układu zarządzającego pamięcią (MMU) 16082. System ma dwa tryby pracy: użytkowy i nadzorujący (ang. supervisor state). Tryb nadzorujący umożliwia wykonywanie wszystkich rozkazów na dowolnych argumentach, podczas gdy tryb użytkowy blokuje realizację niektórych rozkazów lub stosowanie pewnych argumentów. Normalnie 32032 pracuje w trybie użytkowym, przechodząc do trybu nadzorującego w wyniku realizacji rozkazów wywołania nadzoru, pułapek (ang. traps) lub przerwań. Przechodząc do trybu nadzorującego program użytkowy przekazuje sterowanie systemowi operacyjnemu — zapobiega w ten sposób potencjalnym uszkodzeniom.

MMU pozwala na realizację pamięci wirtualnej. Pamięć wirtualna i fizyczna są podzielone na strony o określonych formatach. Program, a także odpowiadający mu obszar danych — wprowadzone są przez system operacyjny do pamięci fizycznej. System operacyjny wprowadza również tablicę konwersji adresu wirtualnego na fizyczny. Tablica ta nie tylko dostarcza informację o konwersji adresu, ale również wykazuje obecność strony pamięci wirtualnej w pamięci fizycznej i opisuje stopień zabezpieczenia przed niepożądanym dostępem dla każdej strony pamięci. Czas konwersji adresów wynosi 100 ns, a praktyczny stopień wykorzystania „ukrytej" pamięci wynosi 98%.

Przykłady stosowania

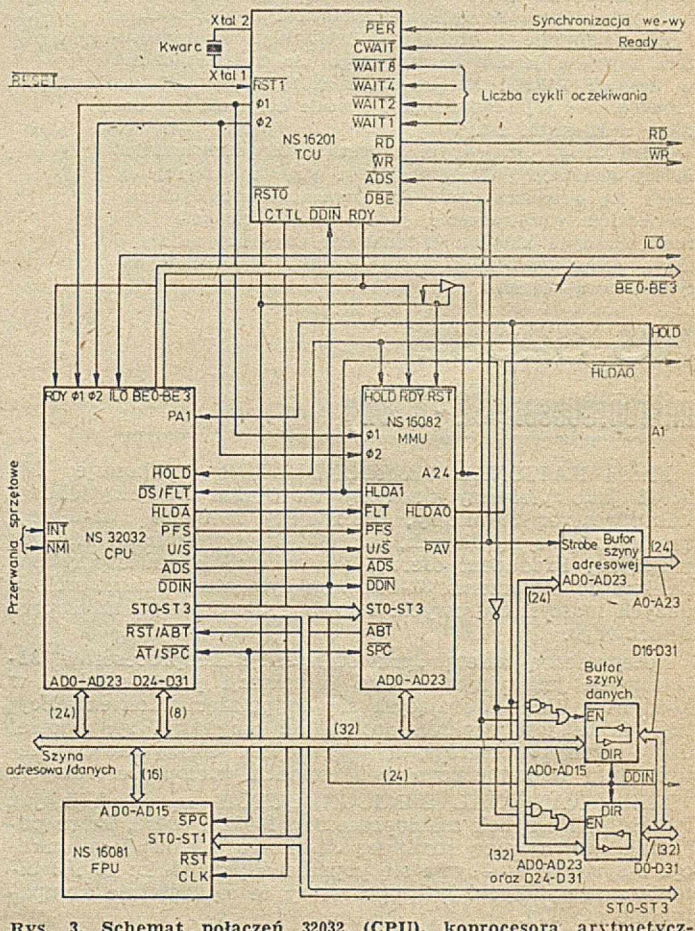
Rysunek 2 przedstawia uproszczony schemat połączeń 32032, 16201 i bloku pamięci „o szerokości" 32 bitów. TCU (16201) jest odpowiedzialny za funkcje zerowania, generowania sygnałów zegarowych i synchronizację pamięci z CPU.

Operacja na szynie rozpoczyna się od podania przez CPU Sygnału ADS (ang. Address Strobe), który służy do demultipleksowania adresu pochodzącego z szyny danych. TCU generuje sygnały odczytu lub zapisu do pamięci, zależnie od stanu sygnału określającego kierunek transmisji danych DDIN/ wysyłanego przez CPU. TCU dostosowuje przebieg operacji do szybkości pamięci i urządzeń peryferyjnych wprowadzające cykle oczekiwania. Gdy CWAIT/ jest aktywny, cykle oczekiwania są wprowadzane aż do momentu przejścia sygnału w stan nieaktywny. Cztery wejścia WAIT/ umożliwiają wprowadzenie od zera do piętnastu cykli oczekiwania. Wejście PER/ umożliwia wprowadzenie pięciu cykli oczekiwania i synchronizację sygnałów odczytu lub zapisu z urządzeniami we-wy.



Rys. 2. Uproszczony schemat połączeń 32032 i układu zegarowego 16201

Stosowanie koprocesorów



Rys. 3. Schemat połączeń 32032 (CPU), koprocesora arytmetycznego (FPU) i układu zarządzania pamięcią (MMU)

Rysunek 3 przedstawia przykład połączenia koprocesorów PPU 16081 i MMU 16082 z układami CPU i TCU. Połączenia sprowadzają się do 16 mniej znaczących bitów szyny danych, linii statusu CPU STO-ST3 i sygnału SPC/. Pozostaje jeszcze doprowadzenie zerowania i zegara do FPU. MMU, ze względu na realizowane funkcje, wymaga ponadto połączeń dodatkowych. Dotyczą one całości szyny adresowej i sygnału ADS/.

LESZEK MADEJA
Warszawa

Obywatelu! Nie wybieraj między klanami. Bądź sobą

μK

Pisaliśmy już o sposobach współpracy z napędami dysków elastycznych. Tym razem ciekawostka, czyli jak współpracować z napędem dysków, który bardziej znany jest z legendy niż z rzeczywistości i którego interfejs przypomina wszystko tylko nie rozwiązania przyjęte na świecie. Mowa tu o produkowanej do niedawna przez Zakłady MERA-KFAP jednostce PLx45D. Pozwala ona na współpracę z dyskietkami jednostronnymi, przy zastosowaniu pojedynczej gęstości, lecz za to ośmiocalowymi, które coraz trudniej spotkać na Zachodzie.

Ale cóż, wyjątkowa waga, jaką przykładają tzw. czynniki decyzyjne do rozwoju dziedziny, która dla zaskakująco dużej liczby państw stała się najważniejszą, powoduje że przysłowie o bezrybiu i raku nadal obowiązuje.

Nie zamieszczamy informacji o układzie sterownika 8271, gdyż jest to już nieco leciwy produkt i opisany został m.in. w kilku kolejnych edycjach katalogów firmy INTEL.

Współpraca z napędem dysków elastycznych PLx45D (1)

Jednostka dysków elastycznych PLx45D zawiera dwie kieszenie oraz dwie głowice, sterowane wspólnym napędem i wspólnymi sygnałami. Można w niej umieścić dwie dyskietki, ale dostęp do nich nie może być jednoczesny.

Do sterowania PLx45D przeznaczone są sygnały:

RUN/ — włączenie napędu dysków i odblokowanie wybranych układów wewnętrznych
DISKR/ — wybór prawej kieszeni dysku, podłączenie sygnałów odczytu-zapisu do tej kieszeni
STEPIN/ — przesunięcie głowicy o jedną ścieżkę w kierunku środka dyskietki
STEPOUT/ — przesunięcie głowicy o jedną ścieżkę na zewnątrz dyskietki
LOAD/ — opuszczenie głowicy odczytu-zapisu
ERSEN/ — włączenie układu „kasowania tunelowego"
WRTEN/ — zezwolenie na zapis
WRDATA/ — zakodowane dane do zapisania

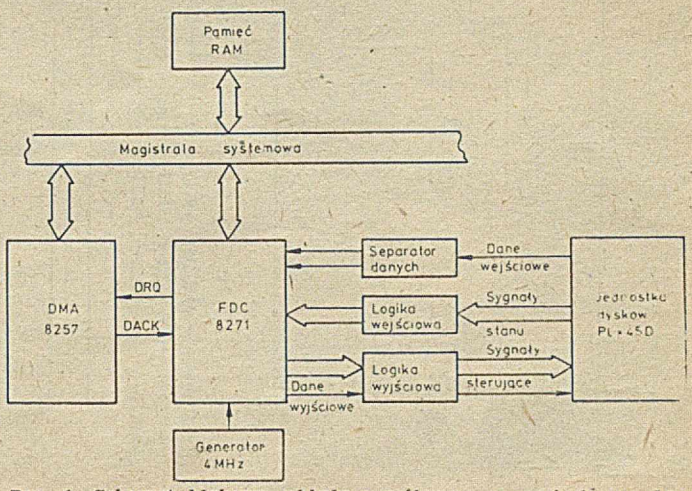
Z jednostki dysków PLx45D wyprowadzone są sygnały:

RDYL/ — gotowość lewej dyskietki
DRYR/ — gotowość prawej dyskietki
INDXL/ — impuls indeksu lewej dyskietki
INDXR/ — impuls indeksu prawej dyskietki
LSDA/ — lewa dyskietka zwrócona stroną A do głowicy
RSDA/ — prawa dyskietka zwrócona stroną A do głowicy
HOME/ — głowica znajduje się w pozycji spoczynkowej nad ścieżką —1
OUTPUT/ — zakodowany sygnał danych, odczytanych z dyskietki
RCLK/ — odtworzony z dyskietki pełny sygnał zegarowy
CDATA/ — dane zegara odczytane z dyskietki
RDATA/ — dane odczytane z dyskietki
Wszystkie sygnały są aktywne gdy sterowane są poziomem niskim.

Z porównania sygnałów jednostki dysków PLx45D oraz układu 8271 wynikają następujące wnioski:

— wykorzystując układ 8271 można wytworzyć wszystkie sygnały sterujące dla PLx45D
— występuje niezgodność między sygnałem **HOME/** i **TR K0/**. Układ 8271 wymaga sygnalizacji, że głowica znajduje się nad ścieżką 0, podczas gdy PLx45D sygnalizuje położenie głowicy nad ścieżką —1
— sygnał **OUTPUT** może być wykorzystany do wytworzenia sygnału **UNSEPARATED DATA/** oraz **DATA WINDOW/**, jednak znacznie mniej zakłóceń mają sygnały **RCLK, CDATA** i **RDATA.**

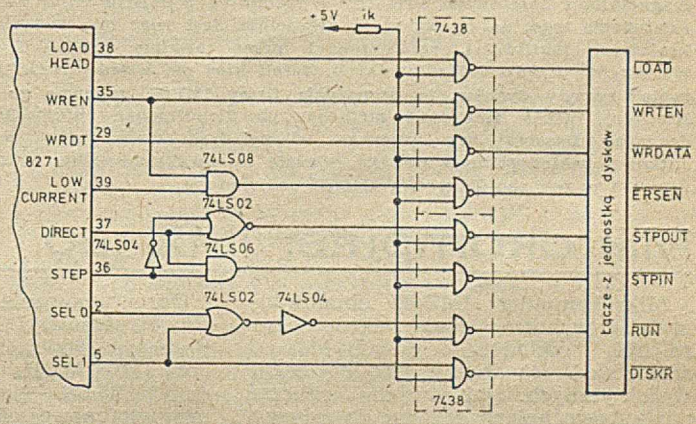
Dołączenie stacji dysków za pośrednictwem układu 8271 przedstawiono w sposób blokowy na rysunku 1.



Rys. 1. Schemat blokowy układu współpracy z pamięcią na dyskach elastycznych z wykorzystaniem kanału DMA

Pomiędzy układem 8271, a jednostką dysków znajdują się trzy zespoły:

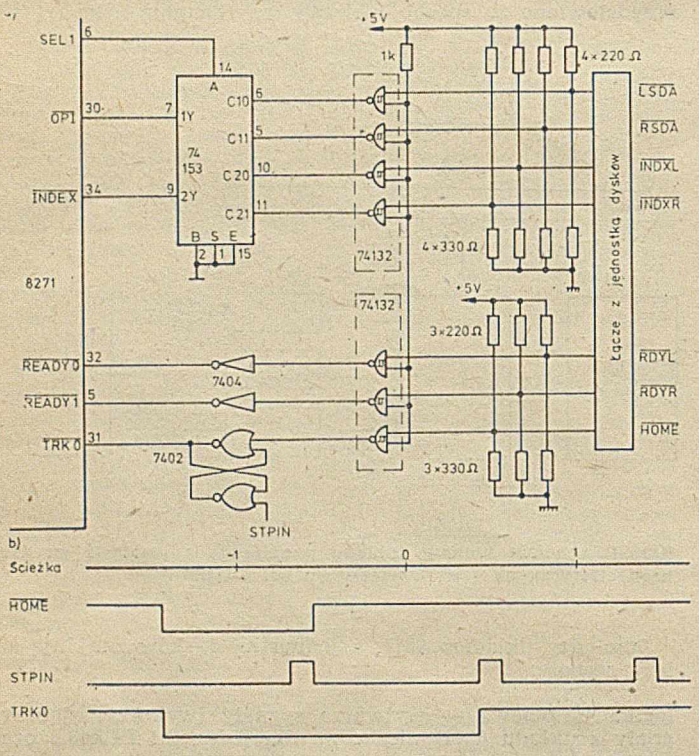
logika wyjściowa — wytwarza sygnały (wykorzystując sygnały z układu 8271) niezbędne dla jednostki PLx45D oraz odpowiednio je wzmacnia
logika wejściowa — wytwarza sygnały niezbędne dla układu sterującego, wykorzystując sygnały przychodzące z jednostki dysków
separator danych — wytwarza sygnały danych dla układu 8271, wykorzystując sygnały odczytu z dysku.



Rys. 2. Logika wyjściowa

Przykładowy schemat zespołu logiki wyjściowej przedstawiono na rysunku 2. Sygnał **LOW CURRENT** należy ziloczynować z **WRDT** dla otrzymania właściwego sygnału **ERSEN.** Wyjście **DIRECT/** zostało wykorzystane do przełączenia sygnału **STEP/** w celu wytworzenia sygnałów **STPIN/** i **STPOUT/**. Sygnał **RUN/** powstał w wyniku zsumowania sygnałów **SEL1/** i **SEL0/**, natomiast sygnał wyboru dysku **DISKR/** odpowiada sygnałowi **SEL1/**.

W celu wyeliminowania zakłóceń toru przesyłowego sygnały z jednostki dysków wprowadzane są przez bramki Schmitta, na wejściu których umieszczono oporniki dopasowujące. Ponieważ układ 8271 ma tylko jedno wejście **INDEX/**, sygnały **INDXL/** i **INDXR/** muszą być przełączane w taki sposób, aby doprowadzony był sygnał z aktualnie wybranego dysku. W tym celu wykorzystany został sygnał **SEL1/**.



Rys. 3. a) logika wejściowa, b) wytwarzanie sygnału TRACK 0

Sygnały strony dysku **LSDA/** i **RSDA/** zostały wprowadzone na opcjonalne wejście **OPI/**. Podobnie jak w przypadku indeksu są one przełączane. Dzięki temu program obsługi może określić położenie aktualnie wybranego dysku.

Największy problem stanowi wytworzenie sygnału **TRK0/**. Układ 8271 przy odwołaniu do sektora znajdującego się na ścieżce 0 „przesuwa głowicę" do napotkania sygnału **TRK0/** (niezależnie od stanu wewnętrznego licznika ścieżek). Dlatego, sygnał **TRK0/** musi być sterowany podczas każdej operacji na ścieżce 0. Jednostka dysków PLx45D nie może tego zapewnić, gdyż jej sygnał **HOME/** jest aktywny tylko wtedy, gdy głowica znajduje się na zewnątrz od ścieżki 0.

Rozwiązanie przedstawione na rysunku 3 wymaga specjalnej obsługi w programie sterującym. Musi on kontrolować czy operacja dotyczy ścieżki 0. Jeżeli tak, to poprzez odczyt rejestru numeru ścieżki program sprawdza czy głowica nie jest już ustawiona na ścieżce 0. Jeżeli konieczne jest przesunięcie głowicy, program wysyła rozkaz przesunięcia na ścieżkę 0, co powoduje, że głowica będzie się przesuwać aż do pojawienia się sygnału **HOME/** i znajdzie się na ścieżce —1. Rozkaz przesunięcia głowicy o jedną ścieżkę sprowadzi ją na ścieżkę 0. Należy jeszcze zaktualizować wewnętrzny licznik ścieżek układu, przez wykonanie rozkazu zapisu do tego rejestru.

W drugiej części artykułu omówione zostaną różne układy wykorzystywane przy wprowadzaniu danych odczytywanych z dyskietki. Przedstawiony zostanie również szkic procedur programu obsługi pozwalającego zrezygnować ze stosowania układu DMA.

(cdn)

ADAM PLUTA
PIE — Warszawa

Prezentujemy kolejny mikrokomputer polskiej produkcji. Jest to dość szczególna konstrukcja: trudno oprzeć się wrażeniu, że IMZ-80 został zaprojektowany jako system wykorzystujący w charakterze pamięci masowej — dyski elastyczne (niewielkie oprogramowanie rezydencyjne). Propozycja wykorzystania magnetofonu kasetowego świadczy o rzetelnym podejściu firmy — nie obiecuje załatwienia tego, na co szanse są niewielkie. Jest więc to rozwiązanie na miarę krajowych możliwości, ograniczonych przez rozmiary produkcji państwowych wytwórców urządzeń peryferyjnych. Jeżeli ktoś ma własne „wejścia" i „załatwi sobie" napędy dysków, to propozycja firmy IMPOL zapewne go zainteresuje. Nie musi wtedy marnować pieniędzy na niepotrzebne oprogramowanie rezydencyjne. Desperat bez szansy na własne dyski może jednak wpisać w dodatkowy pakiet pamięci EPROM właśnie to oprogramowanie, które najczęściej używa i w ten sposób trochę ułatwić sobie pracę.

Mikrokomputer IMZ-80

Mikrokomputer IMZ-80 zbudowany został z wykorzystaniem mikroprocesora Z80 i układów o dużym stopniu scalenia. Przeznaczony jest do celów dydaktycznych, biurowych, obliczeń naukowo-technicznych oraz do zabawy.

Wersja podstawowa wyposażona jest w pamięć operacyjną 64 KB (4 KB zajmuje pamięć obrazu i obszar roboczy klawiatury), klawiaturę typu QWERTY, układ współpracy z magnetofonem, układ sterujący monitorem ekranowym i modulator pozwalający na przyłączenie komputera do telewizyjnego wejścia antenowego. IMZ-80 zawiera także zegar i generator sygnałów akustycznych.

Oprogramowanie dla wersji podstawowej stanowi LOADER, umieszczony w pamięci stałej (tzw. SHADOW ROM) oraz MONITOR, BASIC i MACRO-ASSEMBLER — wczytywane z kasety magnetofonowej. MACROASSEMBLER zawiera EDITOR, KOMPILATOR i DEBUGGER programów, pisanych w języku ASSEMBLER Z80.

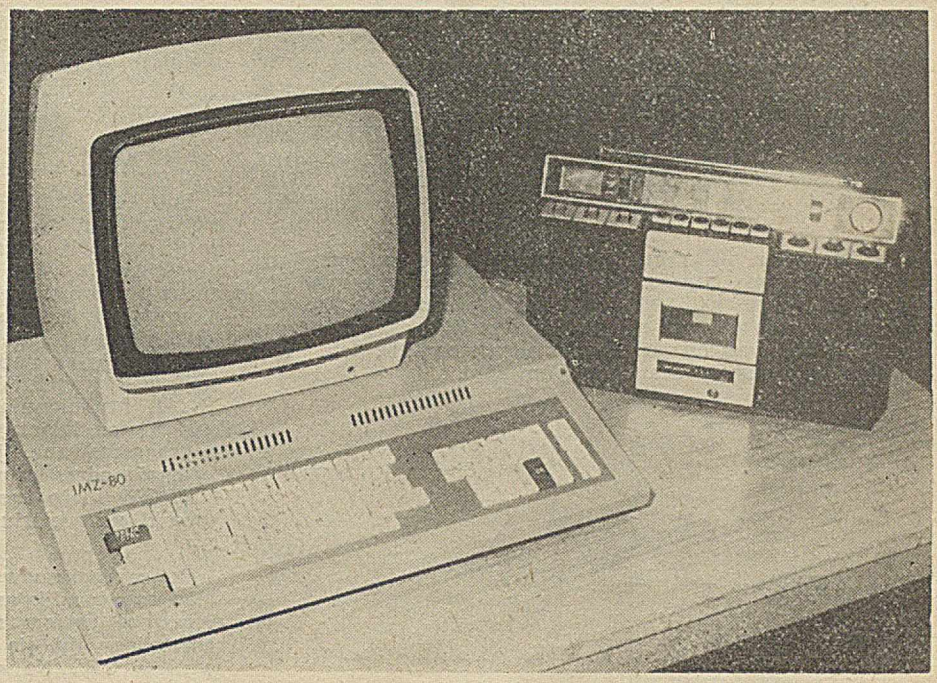
Rozbudowę konfiguracji mikrokomputera umożliwia tzw. ekspander. Jest to moduł buforujący sygnały magistrali mikroprocesora i pozwalający na umieszczenie we wbudowanej kasecie — do sześciu dodatkowych modułów. Zestaw oferowanych (lub przygotowywanych) przez producenta modułów zawiera: sprzęg dla drukarki DZM 180, sprzęg dla pamięci kasetowej PK3, moduł pamięci stałej EPROM — 16 KB, programator pamięci EPROM dla typów 2716 i 2732, szeregowy sprzęg komunikacyjny V24 i układ współpracy z napędami dysków elastycznych 5,25" i 8". Jako napęd dysków 8" przewidziano PLx45D (prod. MERA-KFAP) natomiast jako napędy 5,25" mogą być wykorzystane K5600.10 (prod. ROBOTRON, NRD).

Wyposażenie IMZ-80 w pamięć na dyskach lub minidyskach elastycznych przekształca go w „rasowy" mikrokomputer przydatny w zastosowaniach profesjonalnych. Pozwala to również na wykorzystywanie bogatego oprogramowania: systemu operacyjnego IMPS (funkcjonalny odpowiednik znanego systemu CP/M), interpretera języka BASIC (MBASIC), kompilatorów języków. BASIC 80, FORTRAN 80, PASCAL/M, makroasemblerów itp.

Podstawowymi urządzeniami wejścia-wyjścia są klawiatura i monitor ekranowy. Klawiatura ma wydzielone pola alfanumeryczne, numeryczne i funkcyjne, w tym klawisze sterowania ruchem kursora. Trochę kłopotu może sprawić użytkownikowi oswojenie się z podwójnym znaczeniem klawiszy sterowania ruchem kursora: ten sam klawisz użyty bez SHIFT-u przesuwa

IMZ-80

CPU: Z80 (ZILOG)
RAM: 64 KB
(w tym 4KB — pamięć obrazowa)
Sprzęg magnetofonu
Zegar
Sterownik wejścia antenowego OTV
Generator akustyczny
Oprogramowanie:
MONITOR, BASIC, MACROASSEMBLER
Opcje:
— EPROM 16 KB
— sprzęg drukarki DZM 180
— sprzęg pamięci kasetowej PK3
— sprzęg V24
— sprzęg dysków lub minidysków elastycznych i system operacyjny IMPS
— programator EPROM
Cena wersji podstawowej — około 480 tys. zł.
Czas realizacji zamówienia — ok. pół roku.
Producent: Przedsiębiorstwo Polonijno-Zagraniczne IMPOL II, Warszawa, ul. Filona 16 (Mokotów) tel. 43-03-84, 43-75-66, 43-93-41, teleks 817-218



kursor w prawo, a z SHIFT-em — w lewo.

Na ekranie może być wyświetlanych 25 wierszy po 40 znaków. Do tego celu można wykorzystać monitor (np. NEPTUN 156 — UNIMOR) lub standardowy odbiornik telewizyjny. Zestaw wyświetlanych znaków zawiera 96 znaków ASCII i 64 znaki semigraficzne.

Mikrokomputer wraz z dodatkowymi modułami, umieszczonymi w wewnętrznej kasecie zasilany jest z wbudowanego zasilacza impulsowego o wysokiej sprawności, dzięki czemu możliwa jest długotrwała praca bez wewnętrznej i zewnętrznej wentylacji. Moduły zewnętrzne (np. pamięć dyskowa) wymagają zewnętrznego zasilania, czego nie należy chyba uważać za dużą wadę mikrokomputera. Natomiast niewątpliwą zaletą IMZ-a jest łatwość jego rozbudowy.

Pierwsze egzemplarze IMZ-80 przechodzą „chrzest bojowy" w ZETO — Olsztyn, Klinice Laryngologicznej AM w Lublinie, Instytucie Meteorologii i Gospodarki Wodnej oraz w kilku innych instytutach badawczych. Użytkownicy opracowują własne oprogramowanie użytkowe, dostosowane do indywidualnych potrzeb. W zastosowaniach przeważają jednak obliczenia inżynierskie i naukowe, w tym zbieranie i przetwarzanie danych statystycznych.

Programy testowe, proponowane w mikroKLANIE 3/84, IMZ-80 wykonuje z następującymi wynikami: 1 — ok. 1 s, 2 — ok. 8 s, 3 — ok. 15 s, 4 — ok. 22 s, 5 — ok. 24 s, 6 — ok. 35 s, 7 — ok. 52 s.
Suma czasów wynosi ok. 157 sekund. Pomiary zostały wykonane przy użyciu zegarka z sekundnikiem.

WIESŁAW KASKA
IMPOL II

EO/1504/K/84

Żaden budowany przez hobbystę, ba, nawet produkowany seryjnie (w krajowych warunkach) system mikroprocesorowy nie może się obejść bez pamięci EPROM. W krajach zachodnich ROM-y opłaca się wykorzystywać dopiero przy seriach ponad tysiąc egzemplarzy. U nas — przy jeszcze większych.

Osobną klasą zastosowań są „domowe" modyfikacje oprogramowania fabrycznych mikrokomputerów. I tu pamięć EPROM oddaje nieocenione usługi.

CEMI nie opanowało jeszcze technologii układów EPROM, trzeba więc bazować na układach importowanych. Odpowiednik 2716 produkowany jest w ZSRR i choć nie ma to większego znaczenia dla hobbystów (nie spotyka się go w sklepach), to jednak nie pozostaje obojętne przy konstrukcjach profesjonalnych. W systemach dyskowych 2716 wystarcza do przechowania rezydencyjnej części oprogramowania.

Opisywana poniżej realizacja programatora jest na tyle prosta, że jej konstrukcja i oprogramowanie nie powinno stanowić problemu nawet dla początkujących.

Programator pamięci EPROM 2716

Na rysunku 1 przedstawiono schemat ideowy rozwiązania pozwalającego programować pamięci EPROM (ang. Erasable Read Only Memory) typu 2716. W rozwiązaniu wykorzystano układ programowanych portów 8255 (polski odpowiednik: MCY 7855). Może on współpracować z mikrokomputerami, w których jako jednostkę centralną zastosowano mikroprocesor 8080, Z80 lub nawet 8086. Bez większych problemów można więc wykorzystać opisywany układ programatora dla większości popularnych mikrokomputerów.

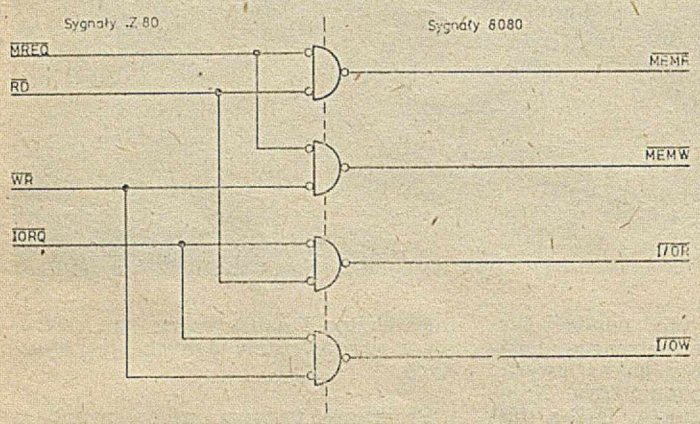
Przedstawione rozwiązanie można również przystosować do programowania innych typów pamięci EPROM. Jednak, mimo zbliżonej konfiguracji wyprowadzeń, nie można zbudować uniwersalnego programatora, w którym nie zachodziłaby potrzeba przełączania podłączeń. Dla pamięci o większej pojemności (np. 8K × 8) zamiast gniazda z 24 wyprowadzeniami trzeba zastosować 23-wyprowadzeniowe.

Podłączenie do mikrokomputera

Współpraca programatora z systemem mikroprocesorowym wymaga umieszczenia układu 8255 w określonym miejscu obszaru adresowego przestrzeni we-wy lub pamięci. Zazwyczaj wygodniej jest wykorzystać przestrzeń we-wy, gdyż pozwala to na potencjalne wykorzystanie pełnych 64 KB pamięci. Wariant ten przedstawiono na rysunku. Podłączenie programatora w przestrzeni adresowej

pamięci pozwala jednak na napisanie procedur obsługi w języku BASIC, z wykorzystaniem instrukcji PEEK i POKE. Jeżeli więc mikrokomputer nie wykorzystuje pełnych 64 KB przestrzeni adresowej pamięci, warto zmodyfikować przedstawione rozwiązanie. W tym celu należy zastąpić sygnały I/OW/ i I/OR/ sygnałami MEMW/ i MEMR/. Konieczne może się też okazać rozbudowanie dekodera adresów, gdyż do adresowania pamięci wykorzystywane jest 16 linii, a nie 8 — jak w przypadku przestrzeni adresowej we-wy.

Przedstawione na rysunku 1 rozwiązanie wykorzystuje sygnały sterujące generowane w systemach wykorzystujących mikroprocesor 8080 jako jednostkę centralną. Na rysunku 2 przedstawiono sposób konwersji sygnałów sterujących, generowanych przez mikroprocesor Z80, tak aby możliwa była współpraca z układem 8255.



Rys. 2. Konwersja sygnałów sterujących mikroprocesora

W przedstawionym rozwiązaniu dla wydzielenia obszaru adresowego we-wy, w którym umieszczony jest układ 8255, wykorzystano dekoder 8205 (krajowy odpowiednik: 74S405). Dla uzyskania żądanego podobszaru (układ 8255 wykorzystuje cztery komórki adresowe) należy dobrać odpowiednie połączenie w zaznaczonych na schemacie punktach. Linią przerywaną pokazano przykładowe połączenia przydzielające adresy 4...7. Linie adresowe A2, A3 i A4 określają wyjście, na którym powinien pojawić się stan aktywny, gdy na wejściu G1 wymuszony jest poziom wysoki, a na wejściach G2A i G2B — poziom niski.

Układ 2716

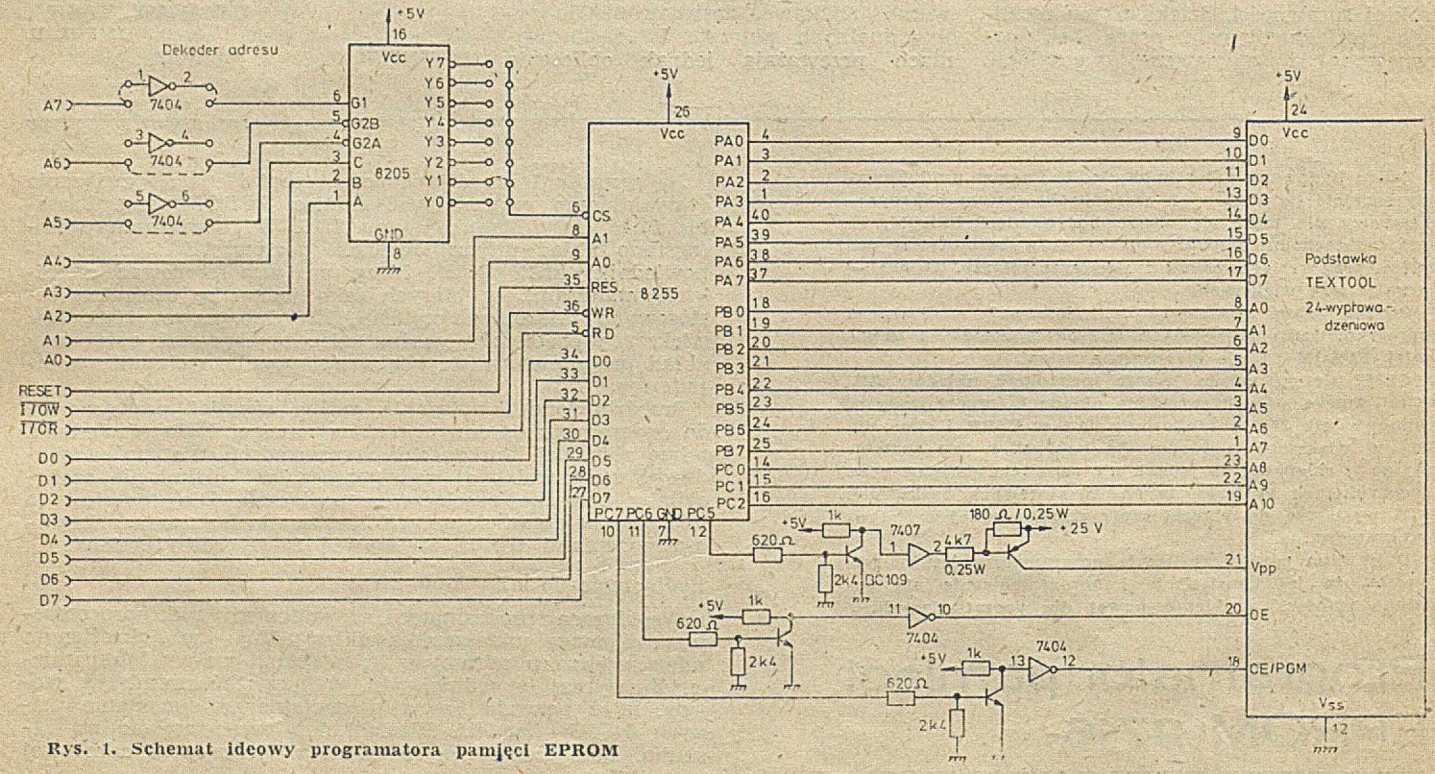
Pamięć EPROM zorganizowana jest w matrycę 2043 jednobajtowych komórek. Dla odczytania żądanej komórki należy wymusić odpowiedni adres na liniach A_0...A_{10}, a na wejście CE/ i OE/ podać stan logiczny "0". Przy odczycie niezaprogramowanej pamięci otrzymujemy kod FF_H. Programowanie polega na wymuszeniu przejścia żądanych bitów w stan logiczny "0". W tym celu na wejście V_{pp} należy przyłożyć napięcie +25 V i wymusić poziom wysoki na wejściu OE/. W trybie programowania wyprowadzenia D_7...D_0 funkcjonują jako wejścia, na których należy wymusić stan logiczny "0" dla tych bitów, które mają zostać zaprogramowane. W chwili gdy na liniach A_0...A_{10} ustabilizuje się adres programowanego bajtu pamięci, na wejście CE/PGM może zostać podany impuls programujący. Stan wysoki na wejściu CE/PGM powinien trwać od 45 do 55 ms.

Po zaprogramowaniu określonego bajtu w pamięci, należy go zweryfikować wykonując odczyt. W tym celu należy utrzymać poziom niski na wejściu CE/PGM (impuls programujący już się skończył) i podać stan logiczny "0" na wejście OE/. Na wejściu V_{pp} można pozostawić napięcie +25 V.

Bity, które w trakcie programowania pozostawiono w stanie logicznym "1", można „doprogramować", powtarzając opisaną procedurę. Bity, które zostały zaprogramowane, nie mogą już zostać zmienione (wrócić do stanu logicznego "1") inaczej niż przez wykasowanie całej pamięci przez naświetlanie promieniowaniem ultrafioletowym.

Program obsługi

Układ 8255 powinien zostać zaprogramowany do pracy w trybie 0. Porty B i C spełniają rolę wyjścia, natomiast port A programowany jest jako wejście, gdy ma być odczytywana informacja z pamięci, i jako wyjście, gdy dane



Rys. 1. Schemat ideowy programatora pamięci EPROM

będą wpisywane do pamięci. Wynika z tego, że między programowaniem a weryfikacją EPROM-u układ 8255 musi zostać przeprogramowany. Ponieważ po zaprogramowaniu układu 8255 w trybie 0 wszystkie wyjścia przechodzą w stan logiczny "0", nie może on być wykorzystywany jako stan aktywny. Dla odróżnienia stanu logicznego "1" od stanu dużej impedancji, wyjścia sterujące zostały zbuforowane kluczami tranzystorowymi. W stanie dużej impedancji baza tranzystora ściągana jest do masy przez rezystor 2,4k, natomiast stan logiczny "1" wysterowuje bazę tranzystora wprowadzając go w stan nasycenia.

Przyłączenie napięcia programującego +25 V następuje w wyniku podania stanu logicznego "1" na wyjście PC5. Wyjścia PC6 i PC7 sterują wejściami OE/ i CE/PRGM.

Stosunkowo długi czas trwania impulsu programującego umożliwia napisanie programu realizującego wcześniej opisany algorytm w języku wyższego poziomu (np. BASIC).

Uwagi konstrukcyjne

Napięcie +25 V zazwyczaj nie jest dostępne w układach mikrokomputerowych. Dlatego wygodnie jest przewidzieć na płytce programatora specjalne zaciski pozwalające na przyłączenie zewnętrznego zasilacza. Innym sposobem może być zbudowanie specjalnej przetwornicy napięcia z +5 V na +25 V. Obciążalność prądowa przetwornicy może nie przekraczać 30 mA.

W programatorze pamięci należy zastosować specjalną podstawkę produkowaną przez firmę TEXTOOL. Umożliwia ona swobodne włożenie układu, a następnie dociśnięcie blaszek kontaktowych za pomocą specjalnej dźwigienki. „Zwyczajne" podstawki nie są odporne na tak częste wkładanie i wyjmowanie układów i stosunkowo szybko niektóre wyprowadzenia przestają kontaktować. Ostatnio na rynku krajowym pojawiły się podstawki bardzo zbliżone do podstawek firmy TEXTOOL produkowane przez warszawskiego rzemieślnika.

Różnice w rozmieszczeniu wyprowadzeń dla różnych pamięci EPROM

Nr wyprowadzenia	2516	2532	2716	2732
18	PD/PROG	A_{11}	CE/PROG	CE/
20	CE	PD/PROG	OE/	OE/V_{pp}
21	V_{pp}	V_{pp}	V_{pp}	A_{11}

W tabeli zestawiono różnice w rozmieszczeniu wyprowadzeń dla różnych typów pamięci EPROM. Algorytm programowania jest identyczny, zatem przez zastosowanie odpowiedniego przełącznika na płytce programatora można rozszerzyć asortyment układów.

AJP

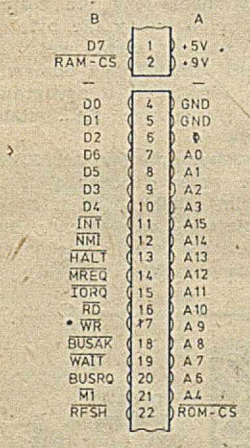
Mimo zaprzestania produkcji ZX81 przez firmę SINCLAIR, ten prosty mikrokomputerek dostępny jest jeszcze w wielu sklepach na Zachodzie i ze względu na niską cenę ciągle przybywa jego użytkowników w naszym kraju. Mimo że nie jestem przekonany czy jest to celowy dla kogokolwiek zakup (prymitywna konstrukcja, niewielkie możliwości) to nie będę nikogo piętnował za marnowanie własnych pieniędzy. Lepszy taki mikrokomputer niż żaden.

Odpowiadając na rosnące zainteresowanie przygotowujemy w mikroKLANIE jeszcze kilka publikacji, tym razem o zagadnieniach sprzętowych związanych z ZX81.

Schemat ZX81

Na rysunku 1 przedstawiono schemat mikrokomputera ZX81. Uwzględnia on fakt, że produkowane były dwie wersje ZX81 różniące się rozmiarem wbudowanej pamięci RAM. Najczęściej spotykana jest wersja zawierająca dwa układy pamięci statycznej 2114 co daje w sumie 1 KB pamięci RAM. Po odliczeniu obszaru wykorzystywanego przez komputer dla przechowywania informacji niezbędnych przy funkcjonowaniu oprogramowania firmowego dla użytkownika pozostaje ok. 0,7 KB. W drugiej wersji wykorzystywano pamięć dynamiczną 4118 lub 4816.

Mikrokomputer zbudowany został z wykorzystaniem jednego z najpopularniejszych mikroprocesorów ośmiobitowych: Z80 (zastosowano szybszą wersję oznaczoną literą A). Częstotliwość zegara doprowadzonego do Z80 wynosi 3,25 MHz. Warto zwrócić uwagę, że zastosowano tu dość niekonwencjonalne rozwiązanie, wprowadzając możliwość chwilowego wstrzymywania zegara przez układ **SCL** (ang. Sinclair Computer Logical). Jest to stosowane zamiast wprowadzania Z80 w cykl oczekiwania, gdy z jakichś względów nie może on skomunikować się z zasobami w „normalnym" czasie. Użytkownikowi pozostawiono jednak możliwość zastosowania konwencjonalnego rozwiązania w przypadku dobudowywania rozszerzeń sprzętowych. W tym



Rys. 2. Topologia wyprowadzeń na złączu użytkownika w ZX81

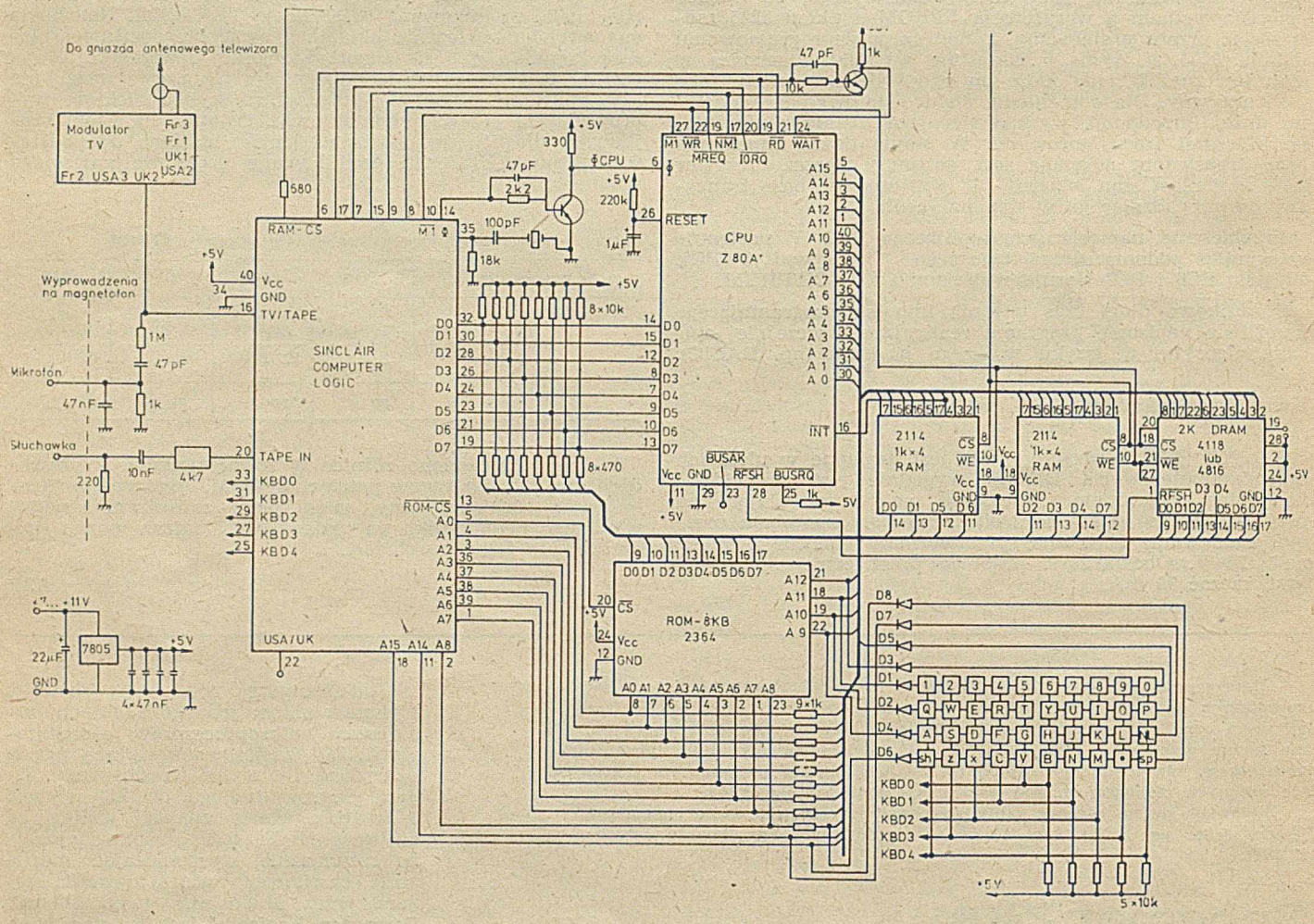
mikroKLAN usamodzielniony?

Kolegium redakcyjne INFORMATYKI uznało, że mikroKLAN należy wydzielić jako osobne pismo. To jednak dopiero początek; przed nami miesiące (miejmy nadzieję — tylko miesiące) żmudnego drążenia ścieżek w biurokratycznym monolicie. Optymiści twierdzą, że uda się zacząć od połowy przyszłego roku. Pesymiści, że nigdy. Optymistów jest nieco więcej, stąd też pierwsze rozmowy o nakładzie. Niektórzy twierdzą, że zainteresowanych mikroinformatyką jest najwyżej dwadzieścia tysięcy. Ja zaś uważam, że dobrze robione pismo może mieć 100 tys. nakładu i rozchodzić się będzie w kilka dni. Tym, którzy taką tezę poprą dobrym uzasadnieniem, nadesłanym na nasze ręce, obiecuję pomoc w uzyskaniu prenumeraty na nowe pismo.

Pozostaje jeszcze pytanie, co powinno się znaleźć w „dobrze robionym" piśmie mikroKLAN. Tu również liczymy na opinie Czytelników. W roboczych założeniach pismo rozdzielamy mniej więcej na pół między „sprzętowców" i „programowców". Znajdzie się miejsce także dla początkujących, którym trzeba wyjaśnić rzeczy podstawowe. Specjalne miejsce rezerwujemy na wymianę doświadczeń między użytkownikami („tips and tricks"). Powinno znaleźć się również trochę miejsca na opis zagranicznych i krajowych nowości, a także bieżących wydarzeń.

Wszelkie propozycje można nadsyłać już od zaraz. Także propozycję własnej osoby na redaktora bądź współpracownika samodzielnego mikroKLANU.

AJP



Rys. 1. Schemat ideowy mikrokomputera ZX81 dla wersji z 1 KBi 2 KB pamięci RAM

celu można wykorzystać wyprowadzony na złącze użytkownika sygnał **WAIT/**. Sygnał ten może okazać się wyjątkowo przydatny przy uruchamianiu własnych rozszerzeń sprzętowych. W czasie gdy jest on aktywny, na liniach adresowych (a przy operacji zapisu — również liniach danych) utrzymane są sygnały sterujące.

Wejścia **WAIT/** nie można jednak wykorzystywać gdy komputer pracuje z przystawką rozszerzającą pamięć, zbudowaną na układach pamięci dynamicznej. W czasie cykli oczekiwania (wymuszonych przez niski poziom na wejściu **WAIT/**) nie jest generowany sygnał **RFSH/** i zawartość pamięci dynamicznych nie jest odświeżana.

Na rysunku 2 przedstawiono topologię złącza użytkownika.

AJP

- publikacje o APPLE II pozwoliły odnaleźć się porozrzucanym po kraju użytkownikom tego mikrokomputera. Powstał — pierwszy w Polsce — Klub Użytkowników APPLE. Informacje o klubie i jego działalności można uzyskać kontaktując się z Piotrem Tymochowiczem, tel. domowy: 35 77 60 lub listownie: 01-950 Warszawa, ul. Renesansowa 5 m 10.
- Klub ABAKUS uzyskał osobowość prawną. Dzięki temu jego członkowie mogą już legalnie płacić składki w wysokości 100 zł miesięcznie. Rodzice próbujący zapisać swoje pociechy „na komputery" dziwili się tak niską opłatą, prezes klubu zapowiedział więc jej rychłe podniesienie.
- Pod koniec listopada odbyło się pierwsze z cyklu spotkań poświęconych edukacji wspomaganej mikrokomputerem. Organizatorem jest klub ABAKUS. Zaproszono przedstawicieli ministerstw: Przemysłu Maszynowego, Oświaty i Wychowania oraz Nauki, Szkolnictwa Wyższego i Techniki. Obecni byli także przedstawiciele firm polonijnych POLBRIT i APINA. W trakcie spotkania rozgorzała dyskusja nad sposobem wyposażenia polskich szkół w mikrokomputery. Recepty nie znaleziono. W trakcie dyskusji okazało się, że:

  — Ministerstwo Oświaty przez trzy miesiące nie udzieliło żadnej odpowiedzi na propozycję POLBRITU (dostarczanie kilkudziesięciu tysięcy ZX SPECTRUM rocznie po obniżonej o połowę cenie), bo... budżet planowany jest tylko raz w roku

  — Ministerstwo NSzWiT również nie ma nic do powiedzenia na powyższą propozycję

  — Ministerstwo Przemysłu Maszynowego nie będzie popierać zakupów mikrokomputerów od firm polonijnych, gdyż nigdy nie wiadomo czy nie zostaną one zlikwidowane, a wtedy nie byłoby komu prowadzić obsługi serwisowej

  — producent MERITUM I odmawia użytkownikom informacji o oprogramowaniu firmowym, tłumacząc się... słabą jego znajomością

  — 300 mikrokomputerów ZX SPECTRUM ofiarowane przez austriacką firmę Reitera polskim szkołom już trafiło na honorowe miejsca... w szafach pancernych; nadzieje ofiarodawcy na eksport oprogramowania do Polski okazały się więc płonne.

— prowadzi Andrzej J. Piotrowski tel. dom. 48-22-85

dokończenie ze str. 10

Ocena opisanych mechanizmów według podanych kryteriów zależy od konkretnego obszaru zastosowań. Przykładowo — dla użytkownika opracowującego program czasu rzeczywistego najistotniejsze są: wpływ na szybkość pracy programu oraz możliwość śledzenia po przerwaniu.

Poszczególne mechanizmy oceniono w skali od 0 do 2 dla każdego kryterium. Z przedstawionego porównania wynika, że najlepszy jest program uruchomieniowy wykorzystujący sprzętową realizację krokowego wykonywania programu.

RAFAŁ ZIELENIEWSKI
Instytut Górnictwa Naftowego i Gazownictwa
Kraków

Autor dziękuje mgr. inż. M. Gondzio za cenne uwagi, które przyczyniły się do udoskonalenia treści artykułu.

LITERATURA

[1] Gill J., Fraser D. A.: Software-based Single-step and Multi-register Breakpoint Facility. Microprocessors and Microsystems, Vol. 7, No 2, pp. 81—84, 1983
[2] Hughes T. P., Sawin D. H.: Breakpoint Design for Debugging Microprocessor Software. Computer Design, Vol. 17, No. 11, pp. 93—107, 1978
[3] Smith M. F.: Debugging with Filtered Traces. Computer Design, Vol. 21, No. 4, pp. 143—146, 1982.

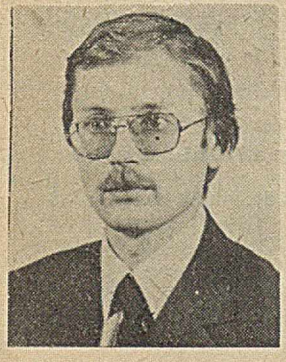
Problemy programowania mikroprocesorów

Na trudności programowania mikroprocesorów wpływa kilka czynników: różnice w zastosowaniach, różnice sprzętowe, różnice w technice wpisywania programu do pamięci i w jego uruchamianiu, a także różnice w oprogramowaniu podstawowym.

Przy przejściu od wykonywania zadań wsadowych do pracy w czasie rzeczywistym wzrasta rola oprogramowania wejścia-wyjścia, a ranga przetwarzania danych ulega obniżeniu. Najczęściej jest to zbieranie i selekcja danych, wyświetlanie wyników i terminowe zarządzanie procesami. Często program użytkowy musi kontrolować upływ czasu. Czasami, programista musi tworzyć fragmenty systemu operacyjnego. Pojawiają się przy tym zagadnienia przerwań asynchronicznych, obsługi urządzeń zewnętrznych na żądanie zadań użytkowych, przydziału procesora zadaniom, łączenia kilku procesorów itp. Zwiększa się znaczenie doboru właściwej bazy danych, pojawiają się nowe typy danych i operatory z nimi związane. Te nowe typy danych często stanowią podzakres zbioru liczb całkowitych. W zależności od sposobu ich upakowania, translacja programu może różnić się efektywnością lub czasem wykonania.

W technice mikroprocesorowej można zetknąć się z wieloma rodzajami pamięci o różnych cechach (na przykład — niezapisywalna) i różnym trybie dostępu. Oprogramowanie mikroprocesorów, w sposób istotniejszy niż programy dużych komputerów, zależy od sprzętu. Częstym wymaganiem, występującym przy pisaniu programów dla mikroprocesorów, są ograniczenia co do czasu wykonywania i pojemności pamięci. Ograniczenia czasowe występują zwykle w programach obsługujących przerwania. Ograniczenia pamięciowe dotyczą zarówno wielkości programu, jak i bazy danych.

Wymienione warunki są źródłem ostrych wymagań co do złożoności, elastyczności i efektywności budowanych kompilatorów. To z kolei powoduje znaczne zwiększenie ich wielkości.



mgr RAFAŁ ZIELENIEWSKI ukończył Wydział Fizyki Uniwersytetu Jagiellońskiego w roku 1979. Od ukończenia studiów pracuje w Instytucie Górnictwa Naftowego i Gazownictwa, gdzie zajmuje się oprogramowaniem mikrokomputerów na potrzeby Okręgowych Dyspozycji Gazu.

Alternatywnym rozwiązaniem jest napisanie kompilatora skrośnego, innym — używanie asemblera. Oba rozwiązania nie są pozbawione wad. Podstawowym mankamentem pierwszego jest czasochłonność pisania takiego kompilatora przy stosunkowo krótkim czasie użytkowania danego mikroprocesora, przeniesienie zaś kompilatora na inny mikroprocesor wymaga wprowadzenia znacznych zmian.

Używanie asemblera jako narzędzia-programisty ma istotne wady. Do podstawowych należy żmudne programowanie, ze względu na uzależnienie notacji od indywidualnego dla danego mikroprocesora repertuaru rozkazów. Po wtóre — brakuje konstrukcji językowych organizujących przebieg wykonywania programu, jak: instrukcje warunkowe, powtarzania, wyboru. Brakuje też ułatwień w dostępie do złożonych struktur danych. Ponadto, nie ma wbudowanego na stałe kodu sprawdzającego poprawność wykonywanego programu (przykładowo — kompilator języka PASCAL na CDC 6000 opcjonalnie generuje kod sprawdzający czy wynik mieści się w zadeklarowanym zakresie). Trudność stanowi takie skoordynowanie dużej liczby podprogramów, które mogą w różny sposób przekazywać sobie parametry. W rezultacie powiększa się znacznie możliwość popełnienia błędu, który jest o wiele trudniej poprawić. Poprawienie błędu w jednym miejscu powoduje czasem błąd w innym, ze względu na wystąpienie skutków ubocznych, na przykład — zmiany rejestru lub wskaźnika, który może wydawać się nieistotny. Dodatkowo, o ile dla programowania w językach wysokiego poziomu ustalono pewne wzorcowe metody postępowania, to dla języków asemblera literatura jest uboższa. Część rad dla programistów sprowadza się do sakramentalnego stwierdzenia „unikaj instrukcji GOTO". Jednakże znajomość wymienionych wad jest korzystna w przypadku konieczności napisania asemblera.

PROJEKT JĘZYKA

Przystępując do konstrukcji asemblera oraz odpowiedniego języka programowania mikroprocesora chcielibyśmy, aby spełniały one takie oto warunki:
— wydruk programu powinien być czytelny i ułatwiać wprowadzenie poprawek
— należy zapewnić możliwość dowolnego stosowania wszelkich „chwytów" programistycznych
— należy umożliwić zachowanie zasad programowania strukturalnego
— należy zagwarantować poprawność semantyczną programu.
Są to oczywiście żądania dość obszerne i niemożliwe do zrealizowania w całości, szczególnie na pojedynczym mikroprocesorze. Dlatego zakłada się utworzenie asemblera skrośnego dla mikroprocesora INTEL 8080.

Zwiększenie czytelności wydruku można osiągnąć przez zastąpienie skrótów mnemotechnicznych dla oznaczania rozkazów, co przedstawiono poniżej.

lxi n,name	n:=name	**adc** r	a:=a+.r
dad n	hl:=hl+n	**sub** r	a:=a—r
stax n	m[n]:=a	**sbb** r	a:=a—.r
ldax n	a:=m[n]	**ana** r	a:=a∧r
shld name	name:=hl	**xra** r	a:=a÷r
lhld name	hl:=name	**ora** r	a:=a∨r
sta name	name:=a	**cmp** r	a:=a?r
lda name	a:=name	**adi** c	a:=a+c
inx n	n:=n+1	**aci** c	a:=a+.c
dcx n	n:=n—1	**sui** c	a:=a—c
inr r	r:=r+1	**sbi** c	a:=a—.c
dcr r	r:=r—1	**ani** c	a:=a∧c
mvi r,c	r:=c	**xri** c	a:=a÷c
rlc	a:=a<—1	**ori** c	a:=a∨c
rrc	a:=a—>1	**opi** c	a:=a?c
ral	a:=a<—.1	**pop** n	n:=top
rar	a:=—>.1	**pchl**	pc:=hl
daa	a:=daa a	**sphl**	sp:=hl
cma	a:=¬a	**xthl**	hl:=: top
stc	ca:=1	**xchg**	hl:=: de
cmc	ca:=¬ca	**push** n	top:=n
mov r,s	r:=s	**jmp** name	**goto** name
add r	a:=a+r		

Rozkazy warunkowe można zastąpić jednolitym odpowiednikiem:

rw	**if** w **then** ret
jw name	**if** w **then goto** name
cw name	**if** w **then call** name

Pozostałe rozkazy nie działające na rejestry:

nop, hlt, di, ei

lub mające czytelne nazwy:

ret, call, out, in, rst

można pozostawić bez zmian.

Powyższy język odpowiada klasycznemu, prostemu asemblerowi tłumaczącemu jedną linię tekstu źródłowego na jeden rozkaz maszynowy. Celowe jest rozszerzenie podanego schematu przez wbudowanie kilku makroinstrukcji, np.:

n:=n+3
a:=a<—3

jako ciągu trzech rozkazów **inx** n lub **rlc**, albo:

c:=c+e
b:=b+.dΛ17q

jako następującego ciągu rozkazów wykonywanych od strony lewej do prawej:

mov a,c; **add** e; **mov** c,a;
mov a,b; **adc** d; **ani** 17q; **mov** b,a;

Jednocześnie asembler powinien dopuszczać upraszczanie instrukcji, np.:

m zamiast m[hl]
a—1 zamiast a:=a—1

UWZGLĘDNIENIE ZASAD PROGRAMOWANIA STRUKTURALNEGO

Naczelną zasadą programowania strukturalnego jest podział programu na mniejsze moduły, przy czym:

— treść modułu powinna zawierać całą wiedzę o strukturze danych i procesach
— przepływ danych między modułami powinien być minimalny i dobrze kontrolowany
— liczba wejść i wyjść modułu powinna być jak najmniejsza.

Dodatkowo, jak wynika z praktyki:

— tekst modułu powinien mieścić się na jednej stronie wydruku
— tworzony kod powinien być wykonywany sekwencyjnie najlepiej bez skoków
— treść powinna być jednoznaczna, bez wywołań pośrednich i samomodyfikacji.

Realizacja postulatu zgromadzenia wiedzy o procesach wiąże się nieodłącznie z zagadnieniem tworzenia rozbudowanych struktur danych. Używanie — na przykład — elementów rekordu pozwala na zmniejszenie liczby błędów i umożliwia asemblerowi kontrolę typu danych. Pożyteczne byłoby zatem wprowadzenie deklaracji występujących np. w PASCALU: const, type, var, value. Przykładowe typy standardowe, które warto wprowadzić, są następujące:

bit 0.. 1
bitx 0..x—1 (x=1,2, ... , 16)
byte —128 .. 127
int —32768 .. 32767
char według życzenia programisty (opcja asemblera)
addr 0..maxaddr (według ustalenia programisty).

Typy strukturalne powinny stanowić tablice i rekordy. Typ addr można uważać za pewne uproszczenie typu wskaźnikowego.

Dla uproszczenia asemblera i ujednoznacznienia kodu, trzeba przyjąć, że mikroprocesor ma pamięć 8-bitową. Asembler będzie zapisywał pola rekordu upakowane kolejno od strony lewej do prawej. Zatem składnię należy uzupełnić o pseudoidentyfikator pola, tzw. filler, np.

```
wskaźnik = packed record jest : bit;
                          filler : bit2 ;
                          wartość : 0..23 ;
           endrecord ;
```

Ponieważ w mikroprocesorach występują zasadniczo dwa rodzaje pamięci RAM i ROM, instrukcja musi rozróżniać, w której pamięci asembler ma zarezerwować obszar — od adresu **minram** czy od adresu **minrom**, np.:

```
ram a, b, c : byte ;
rom x, y, z : int ;
Deklaracja
var s1, s2 : bit8 ;
```

umieszczona wewnątrz podprogramu rezerwuje obszar pamięci nie na stosie lecz na końcu zajętego obszaru pamięci RAM. Ten obszar jest wspólny dla wszystkich podprogramów, dlatego asembler powinien kontrolować, czy zmienne nie są odczytywane bez zdefiniowania wewnątrz podprogramu.

Do zrealizowania postulatów programowania modularnego należy wprowadzić instrukcję:

!proc name

którą można traktować jako komentarz, albowiem nie odpowiada jej żaden kod ani nie powoduje ona zmian w kodzie, aż do nawiasu zamykającego:

!endproc name

Asembler sprawdza wyłącznie istnienie pary

proc i **endproc**, zgodność nazwy, ewentualnie zakończenie podprogramów lokalnych. Instrukcja:

call name
generuje tylko trzy bajty.

Wobec dużej różnorodności sposobów przekazywania parametrów do i z podprogramów, użytkownik musi kodować je sam, a asembler sprawdza tylko zgodność deklaracji i wywołania dla wszystkich zadeklarowanych parametrów.

Przykład

```
!proc mult
!entry deca
!input hl : bit16
!output hl = hl.i×10
!        a:=0
!save b, c, d, e, m
!entry mult
!input hl : bit16
!         a : bit8
!output hl = hl.i×a.i
!         a:=0
!save b, c, d, e, m
deca:a:=10
mult: top:=de        ! top:=de.i
      de:=:hl        ! de:=hl.i
      hl:=0          ! hl + a × de: = a.i × h1.i Λ a ⩾ 0
      aVa
      if z then goto by0     ! h1+a×de:=a.i×h1.i Λ a>0
100p:hl:=hl+de
      a:=a—1         ! h1+a×de:=a.i×h1.i Λ a⩾0
      if ¬z then goto 100p
by0 :de:=top   ! hl + a × de: = a.i × h1.i Λ a = 0 Λ de: = de.i
      ret
!endproc mult
```

Wywołanie podprogramu przedstawionego w przykładzie może być następujące:

```
de:=de+1     !de:= wlko[k] . wskaźnik
a:=m [de]
a:=a<—1 V a —>. 1  ! ca:=wlko [k]. jest Λ a:=wlko[k].
                                                    wartość
hl:=5
if ca then call mult !hl:=wlko[k].wartość×5
```

Użycie zmiennych nielokalnych w podprogramie jest dozwolone tylko po uprzednim deklarowaniu. Dozwolone jest przedefiniowywanie typu ze względu na ograniczenie dostępu, np.:

```
! global
wskaźnik = packed record jest :r: bit ;
                         filler : bit2 ;
                         wartość :w: 0..23 ;
                  endrecord ;
```

gdzie:

:r: zezwala tylko na odczyt
:w: zezwala na odczyt tylko po uprzednim zapisaniu
:rw: zezwala na swobodne korzystanie.

Zapamiętywanie i dostęp do zmiennych lokalnych przechowywanych na stosie jest możliwy tylko przez programistę, ale na podstawie komentarzy asembler powinien sprawdzić, czy stos jest zrównoważony w każdym module i czy odbierane są te zmienne, które są oczekiwane.

INNE ROZSZERZENIA

Myślą przewodnią projektu jest skrócenie czasu pracy programisty nie przez programowanie w języku wysokiego poziomu, ale przez zwiększenie niezawodności tworzonych programów. Oczywiście, obie drogi nie wykluczają się nawzajem.

Proponowany asembler powinien uwzględniać reguły dowodzenia poprawności semantycznej przez sprawdzanie tzw. niezmienników Hoare'a [1]) zadanych przez programistę. Tworzenie niezmienników dla programów mikroprocesorowych wydaje się łatwiejsze, bo z reguły są one mniej złożone i często obejmują algorytmy nienumeryczne, które można opisać formalnie. W przykładowym podprogramie mult zaznaczono te niezmienniki jako komentarze dla asemblera. W zależności od „inteligencji" asemblera, część z nich można pominąć.

Siłę takiego asemblera zwiększyłoby wprowadzenie instrukcji strukturalnych. Instrukcja warunkowa o postaci:

```
if a:=m[hl] <0 then !if wlko[k].jest ≠ 0 then
!     begin
      if aΛ37q = maxtypwartość then
      m:=200q ! jest:=1 Λ wartość:=0
!     else
      m:=m+1    ! jest:=1 Λ wartość:=wartość+1
!     end
```

powinna generować odpowiedni kod i sprawdzać — dla obu możliwych przebiegów — zawartości rejestrów przeznaczonych do późniejszego korzystania. Porównanie zawartości rejestrów ze stanem oczekiwań musi odbywać się automatycznie — bez użycia dodatkowej instrukcji.

Instrukcja strukturalna **for** może również stanowić komentarz dla asemblera, np.:

```
ram z : array [0: 127] of bit8 ;

      hl:= z[127]
      c:= 126
! for c:=126 downto 1 do begin
10 1  :a:=m
      hl:=hl—2
      a:=a+m —>. 1 ! a:=z[c+1] +z[c—1] / 2
      hl:=hl+1
      a:=a+m —>. 1 !a:=a+z[c] / 2
      m:=a
      if c:=c—1 ≠ 0 then goto 10 1
! end
```

W powyższym przykładzie asembler powinien sprawdzić, czy pętla wykona się rzeczywiście 126 razy, czy wartość rejestru hl nie wykroczy poza obszar pamięci zarezerwowany dla zmiennej z, czy w instrukcji o etykiecie 101 w każdym przebiegu wystąpi adres z[c+1], czy fragment programu w nawiasach **begin** i **end** ma zrównoważony stos.

Projektowany język jest nadal językiem niskiego poziomu. Pewnym ułatwieniem mogłaby być możliwość definiowania makroinstrukcji i własnych operatorów, np.:

```
!opdef 'a' , ': = . ' ,r
      a:=r
      aVa
!endopdef
```

Jednakże asembler powinien być czymś więcej niż programem do wytworzenia kodu wynikowego. Dobry asembler powinien współdziałać z programistą w celu łatwego tworzenia poprawnego programu. Ułatwieniem w pracy programisty jest na pewno tablica symboli z wyszczególnieniem miejsc odwołań. Programista musi mieć możliwość poprawienia kilku linii programu źródłowego i — natychmiastowego otrzymania informacji, których układów pamięci EPROM dotyczą zmiany (wraz z wykazem zmienionych bajtów). Celowe jest także sprzężenie asemblera z symulatorem mikroprocesora INTEL 8080, co umożliwia lepsze sprawdzenie kodu przez część tłumaczącą program.

* * *

Autor próbuje opracować przedstawiony asembler i będzie wdzięczny za wszelkie uwagi dotyczące projektu.

[1]) Zainteresowanych tą tematyką odsyłamy do książki Alagića i Arbiba „Projektowanie programów poprawnych i dobrze zbudowanych". WNT, Warszawa, 1932

EO/432/K/84

Rada ds. Zastosowań ŚTO

Sekcja SAPR

Sekcja systemów automatyzacji projektowania SAPR (ros. Sistiemy Awtomatizacji Projekitrowania) działa w ramach Rady ds. Zastosowań Środków Techniki Obliczeniowej jako jeden z jej organów roboczych. Reaktywowanie w ubiegłym roku działalności polskiej części Sekcji SAPR skłania do przedstawienia dotychczasowych wyników współpracy międzynarodowej, a także do wytyczenia przyszłych kierunków działania w nowej formie organizacyjnej.

Prawie dziesięcioletni okres współpracy międzynarodowej w ramach tematyki SAPR oceniany jest na ogół pozytywnie, mimo że liczba wdrożeń istniejących już opracowań Sekcji jest w naszym kraju niewielka. Dotychczasowa działalność Sekcji przyczyniła się do powstania wartościowych opracowań, stanowiła też inspirację dla opracowań własnych oraz zapewniła wymianę doświadczeń pozwalających uniknąć podjęcia rozwiązań nietrafionych. Współpraca stworzyła również możliwość poznania współpartnerów oraz pozwoliła na sprecyzowanie wymagań i warunków wspólnego opracowywania tematów. Warto przy tym zaznaczyć, że problematyka dotycząca automatyzacji prac projektowych, konstrukcyjnych i technologicznych jest realizowana nie tylko w ramach SAPR, ale również w ramach innych form współpracy międzynarodowej.

Udział Polski w realizacji tematyki SAPR nie był duży, liczy się jednak w ogólnym dorobku. W chwili obecnej na 37 realizowanych tematów Polska uczestniczy w 14 tematach, przy czym w 4 tematach jest głównym wykonawcą.

Obecnie obserwuje się w Polsce wyraźną stagnację zastosowań informatyki w projektowaniu i konstruowaniu. Wiąże się to w niektórych przypadkach, jak np. w budownictwie, ze znacznym ograniczaniem inwestycji, w

innych jest skutkiem wprowadzenia nowych mechanizmów gospodarki finansowej, które — powodując wysokie koszty techniki komputerowej związane z amortyzacją drogiego sprzętu — hamują skutecznie rozwój zastosowań. W innych dziedzinach, jak np. w przemyśle maszynowym, stagnacja ta wynika z braku zapotrzebowania spowodowanego drastycznym zmniejszeniem liczby nowych uruchomień, prototypów i technologii.

Ten stan stagnacji w zakresie zastosowania komputerów w projektowaniu i konstruowaniu, choć zapewne ma charakter przejściowy, może się przeciągnąć na najbliższe lata. Jest to zjawisko niepokojące, zwłaszcza wobec gwałtownego rozwoju na świecie tej dziedziny zastosowań. Wynika stąd niewątpliwy wniosek, że opracowania SAPR powinny wyprzedzać obecne potrzeby oraz odpowiadać światowym tendencjom rozwojowym.

Nowe tendencje rozwojowe są w szczególności związane z rozwojem sprzętu. Już obecne doświadczenia wskazują, że przyjęta w początkowej fazie opracowań zasada pracy w trybie wsadowym i zastosowanie dużych komputerów do projektowania i konstruowania nie jest rozwiązaniem najlepszym. Niewątpliwie przyszłość rozwiązań będzie opierać się na trybie konwersacyjnym z zastosowaniem aktywnych urządzeń graficznych. Zasadnicze zmiany należy jednak wiązać z rozwojem techniki mikroprocesorowej, która w zasadniczy sposób ukierunkuje rozwój sprzętu. Należy to uwzględnić w planach SAPR na najbliższą przyszłość.

Technologia układów o wielkim stopniu scalenia oraz rozwój architektury systemów wieloprocesorowych i sieci wielokomputerowych o parametrach użytkowych i niezawodnościowych znacznie przewyższających sprzęt poprzednich generacji, a jednocześnie olbrzymi spadek dotychczasowych cen, czynią bardziej dostępne komputerowe wspomaganie projektowania oparte na lokalnych sieciach mikrokomputerowych, a koncepcja rozłożonego przetwarzania staje się podstawowym kierunkiem dalszego rozwoju. Metody przetwarzania oparte na wielotorowym dostępie do zasobów oraz wykorzystanie równoległości obliczeń numerycznych w systemach wieloprocesorowych wykazują znaczną przewagę nad innymi rozwiązaniami pod względem ekonomiczności, niezawodności i możliwości adaptacji.

Oprócz zmian sprzętu, następuje rozwój oprogramowania zgodnie z potrzebami i wymaganiami projektowania. Obecne wymagania przy projektowaniu konstrukcji oraz procesów technologicznych nie sprowadzają się już do jednowariantowych obliczeń; wymagają zastosowania technik modelowania, symulacji i optymalizacji. Występuje tu konieczność integracji metodologicznej tych technik z technikami identyfikacji, wspólnymi dla analiz obliczeniowych i badań eksperymentalnych, a także zastosowania modeli odzwierciedlających procesy robocze projektowanych maszyn, technologii, obiektów — modeli analogicznych, jakie znajdują zastosowanie w zakresie sterowania.

Uwzględniając dotychczasowe doświadczenia, stan obecny, a także światowe tendencje rozwojowe SAPR, za podstawowe kierunki działania na lata 1985—1990 należy uznać następujące zagadnienia:

- **Opracowanie i wdrożenie typowych stanowisk wspomaganego komputerem projektowania — z ukierunkowaniem na zastosowanie w przemyśle maszynowym (konstrukcja, technologia) oraz w budownictwie (projektowanie i konstruowanie obiektów budownictwa). Proponuje się utworzenie trzech wzorcowych ośrodków:**
  — dla konstrukcji maszyn
  — dla projektowania technologii w przemyśle maszynowym
  — dla projektowania obiektów budowlanych.
  W następnym etapie proponuje się utworzenie 2–3 lokalnych sieci mikroprocesorowych na potrzeby SAPR.
- **Opracowanie oprogramowania systemowego typowych stanowisk wspomaganego komputerem projektowania w zakresie zastosowania graficznych urządzeń we-wy, metod symulacji matematycznej, optymalizacji, uniwersalnych metod obliczeniowych itp.**
- **Opracowanie wymagań dla sprzętu komputerowego na potrzeby SAPR oraz programu jego zastosowania w przemyśle krajowym.**
- **Rozwiązanie problemów metodycznych programowych i organizacyjnych związanych z problematyką współdziałania mini- i mikrokomputerów z dużymi komputerami**
- **Rozwiązanie problemów metodycznych, programowych i organizacyjnych związanych z problematyką budowy baz danych oraz integracją niejednorodnych baz danych (dla projektowania technologii, projektowania i konstruowania wybranych maszyn i zespołów).**
- **Typizacja, unifikacja i normalizacja w zakresie SAPR z uwzględnieniem wymagań RWPG.**
- **Zorganizowanie informacji o wykonanych opracowaniach SAPR oraz ich zastosowaniach, promocja najlepszych opracowań, wymiana informacji o prowadzonych pracach i ich wynikach, organizacja szkolenia, sympozja itp.**
- **Opracowanie technologii i oprogramowania użytkowego dla konstruowania maszyn i ich podzespołów w podstawowych branżach rozwojowych przemysłu maszynowego (maszyny budowlane, rolnicze, obrabiarki).**
- **Opracowanie technologii i oprogramowania użytkowego dla projektowania procesów technologicznych w przemyśle budowy maszyn z uwzględnieniem modelowania i optymalizacji oraz metod i środków technologicznych (systemy obróbcze).**
- **Opracowanie technologii i oprogramowania użytkowego dla projektowania architektonicznego, konstrukcyjnego i instalacyjnego budynków oraz sieci zasilających.**

Powyższe kierunki działania, przyjęte na pierwszym zebraniu krajowej części Sekcji SAPR, są zbieżne z głównymi tematami uzgodnionymi podczas posiedzenia przedstawicieli krajów uczestniczących w pracach sekcji SAPR w maju 1984 na Węgrzech.

W ostatnich kilkunastu latach nastąpił na świecie znaczny wzrost zastosowań techniki komputerowej w zakresie wspomagania prac projektowych i konstrukcyjnych, a także przygotowania technologicznego produkcji oraz sterowania procesami wytwarzania (CAD i CAM). Szczególnie dynamiczny wzrost zastosowań techniki komputerowej w przygotowaniu technicznym produkcji i sterowania procesami wytwarzania obserwuje się ostatnio w dziedzinie przemysłu maszynowego. Automatyzacja tego przemysłu, wprowadzenie elastycznych gniazd i systemów obróbczych, robotyzacja itp. są bezpośrednio związane i uwarunkowane rozwojem SAPR.

Zadania polskiej części Sekcji SAPR nie ograniczają się jedynie do reprezentacji naszego kraju w Sekcji SAPR krajów RWPG. Zadania te polegają również na stworzeniu warunków i atmosfery dla rozwoju własnych opracowań, stymulowania tematyki najbardziej potrzebnej dla naszej gospodarki oraz wykorzystania współpracy międzynarodowej dla wykonania oprogramowania niemożliwego do realizacji własnymi siłami, a także wymiany handlowej naszych opracowań z innymi krajami.

Realizacja tych zadań wymaga:
— stworzenia systemu informacji o opracowanych i dostępnych oraz aktualnie opracowywanych programach i systemach SAPR, celem wykorzystania sprawdzonych rozwiązań oraz wymiany doświadczeń i nowych idei
— stworzenia zasad testowania programów i weryfikacji wyników zapewniających promocję najlepszych rozwiązań
— opracowania zasad wymiany i sprzedaży oprogramowania, w szczególności dla partnerów zagranicznych
— sprecyzowania wymagań dotyczących sprzętu komputerowego dla SAPR oraz opracowania programu zaspokojenia istniejących w tym zakresie potrzeb
— wybór najważniejszych tematów realizowanych dla gospodarki krajowej oraz w ramach współpracy międzynarodowej, a także ich finansowanie w ramach przyznanych środków.

W roku 1984, jako jedno z pierwszych zadań, podjęto inwentaryzację opracowanych i użytkowanych w kraju opracowań z zakresu tematyki SAPR w przemyśle elektromaszynowym, budownictwie i energetyce. Na podstawie tych materiałów zostanie wydany biuletyn informacyjny w pierwszej połowie 1985. Wyniki działalności Sekcji SAPR dadzą na pewno lepsze efekty, jeżeli uda się stworzyć odpowiednią atmosferę do współpracy i współdziałania ośrodków i specjalistów — zarówno opracowujących, jak i użytkowujących oprogramowanie. Wymiana myśli, dorobku i doświadczeń przyczyni się zapewne do należytego ukierunkowania i rozwoju tej ważnej dla naszej gospodarki dziedziny. Podając tę krótką informację, zapraszamy wszystkich zainteresowanych do współpracy.

Sekretariat Sekcji mieści się w **Instytucie Obróbki Skrawaniem, 30-011 Kraków, ul. Wrocławska 37a.**

JERZY SIKORA
Przedstawiciel PRL w Sekcji SARP

Pamięci na dyskach optycznych

Wśród perspektywicznych systemów pamięci zewnętrznej na pierwszy plan wysuwają się obecnie pamięci na dyskach optycznych — tzn. pamięci dyskowe, w których odczyt i zapis danych odbywa się za pomocą laserów.

Takie nośniki informacji — opracowane na początku lat osiemdziesiątych — mają postać dysku (zwykle o średnicy 356 mm) pokrytego cienką warstwą telluru. Dane zapisywane są tu w postaci otworów o mikronowych średnicach, wypalanych na powierzchni nośnika za pomocą lasera. Przy odczycie tak zapisanej informacji inny laser — o mniejszej mocy od wykorzystywanego w procesie zapisu — wykrywa na powierzchni dysku otwory, którym odpowiada mniejszy współczynnik odbicia wiązki światła niż pozostałej powierzchni dysku. Jest to jednak nośnik, na którym informację można zapisać tylko raz, bez jakiejkolwiek możliwości jego regeneracji.

Należy zwrócić uwagę na różnicę pomiędzy dyskami optycznymi dla maszyn cyfrowych i dyskami wizyjnymi (wideodyskami) dla elektroniki konsumpcyjnej. Te ostatnie wytwarzane są metodą kopiowania matrycy wzorcowej i przeznaczone są wyłącznie do odczytu przez użytkownika. Natomiast warstwa robocza dysków optycznych dla maszyn cyfrowych powinna być czuła na naświetlenie wiązką laserową, za pomocą której użytkownik zapisuje informację.

Znane są również dyski do zapisu za pomocą lasera sygnałów dźwiękowych (odtwarzanych na aparaturze wysokiej jakości). Dyski te pozwalają użytkownikowi tylko na odczyt zapisanej przez wytwórcę informacji i — podobnie jak wideodyski — wytwarzane są metodą matrycy wzorcowej.

Dla zapewnienia możliwości przepisywania informacji, od początku lat osiemdziesiątych poszukiwano metod zapisu, które okazałyby się równoważne wypalaniu otworów, ale zapewniałyby możliwość wykorzystania laserów do odczytu i zapisu informacji. Na pokrycie nośnika, zapewniającego możliwość przepisywania informacji, zaproponowano wstępnie powłokę z chlorku srebra lub cienką warstwę amorficznego materiału półprzewodnikowego.

Podstawowe właściwości optycznych pamięci dyskowych są następujące:

• **Łatwość przechowywania dużych zbiorów danych.** Potencjalna pojemność typowej jednostki pamięci mieści się w granicach 1—50 G bajtów, czyli jest ona kilkadziesiąt razy większa od maksymalnej pojemności konwencjonalnej pamięci typu IBM 3380. Przechowanie 10^{15} bitów informacji wymaga 1,3 mln szpul taśmy magnetycznej lub 1000 pakietów dysków optycznych, przy czym do przechowania takiej liczby dysków optycznych potrzebna jest powierzchnia rzędu 30 m², natomiast dla przechowywania wspomnianej liczby szpul taśmy magnetycznej — magazyn o powierzchni ponad 3 ha. Przy porównaniu pamięci na dysku optycznym z konwencjonalnymi mikronośnikami informacji okazało się, że na jednym dysku przechowywać można tyle informacji, ile na taśmie mikrofilmowej długości ok. 55 m. Ponadto, w przeciwieństwie do mikrofilmów, informację zapisaną na dysku optycznym można przesłać kanałami łączności do komputera lub do innej pamięci optycznej.

• **Niski koszt przechowywania informacji.** W przypadku dysków z warstwą chlorku srebra koszt ten ma być ok. trzy razy mniejszy niż na taśmie magnetycznej, który w 1981 roku wynosił średnio ok. 0,02 dol. za 1 M bajt. Przewidywany koszt dla pamięci optycznych będzie o rząd wielkości mniejszy.

• **Możliwość przechowywania zapisanej informacji bez przepisywania 4—5 razy dłużej niż w przypadku taśmy magnetycznej,** której przepisywanie jest niezbędne co dwa lata (optymalnie co sześć miesięcy). Oczekuje się, że optyczne pamięci dyskowe zapewnią możliwość niezawodnego przechowywania danych bez przepisywania przez okres dziesięciu lat.

Wśród firm, które prowadzą prace w dziedzinie optycznych pamięci dyskowych, można znaleźć czołowych wytwórców sprzętu informatycznego, jakich jak: IBM, CDC, XEROX, STORAGE TECHNOLOGY, BURROUGHS, NCR, HEWLETT-PACKARD i WANG LABORATORIES (USA), PHILIPS (Holandia), THOMSON — CSF (Francja) oraz SONY, HITACHI i TOSHIBA (Japonia).

Oprogramowanie umożliwiające wykorzystywanie dyskowych pamięci optycznych w systemach komputerowych opracowują również duże firmy innych branż, jak WESTINGHOUSE i HUGHES AIRCRAFT (USA). Natomiast materiały do produkcji dysków optycznych opracowują firmy 3M, KODAK, DREXLER TECHNOLOGY i inne (USA), a także MATSUSHITA (Japonia).

Badania koniunktury i perspektywy rynkowych pamięci optycznych prowadzą specjalistyczne firmy STRATEGIC (USA) i MACHINTOSH CONSULTANTS (W. Brytania). Analiza firmy STRATEGIC wskazywała, że w pamięciach na dyskach optycznych można uzyskać lepsze efekty ekonomiczne niż w przypadku jakiegokolwiek innego urządzenia elektronicznego, z wyjątkiem oczywiście mikroprocesora. Prognoza firmy MACHINTOSH przewiduje wdrożenie do 1990 roku ponad 6000 systemów informatycznych, realizowanych w oparciu o optyczne pamięci dyskowe.

Po okresie wstępnych badań, podstawowymi kierunkami prac w dziedzinie rozwoju optycznych pamięci dyskowych stały się: udoskonalenie ośrodka pamiętającego, opracowanie tańszych laserów półprzewodnikowych (zamiast gazowych), zwiększenie dokładności ustawiania głowic zapisu-odczytu — w związku z 4—5-krotnym zwiększeniem gęstości poprzecznej zapisu (z poziomu 1000 ścieżek/cal w dyskach magnetycznych), zmniejszenie liczby błędów zapisu i odczytu.

Należy jednak uwzględnić fakt, że czas dostępu do informacji zapisanej na tych dyskach jest przeszło dziesięciokrotnie większy od występującego w dyskach typu Winchester (450 ms dla dysków optycznych obracających się z prędkością 900 obr/min oraz 30—40 ms — z prędkością 3600 obr/min). Podobnie czas zapisu informacji na dysku optycznym jest znacznie większy niż na dysku Winchester.

W latach 1982—1983 prowadzono intensywne prace badawcze. W maju 1983 na Krajowej Konferencji Informatycznej w USA japońska firma NIPPON ELECTRIC CO (NEC) zademonstrowała pamięć na wymiennych dyskach optycznych, dostarczanych na rynek amerykański przez firmę 3M. W pamięci tej niemożliwe jest wymazywanie i ponowny zapis informacji. Jako pierwsze jej zastosowanie przewidziano przechowywanie danych i dokumentów, które w czasie nie zmieniają się i wymagają tylko sukcesywnego uzupełniania (podpisy, odciski palców, numery patentów itp.). Początkowo firma NEC zamierzała sprzedawać te pamięci w cenie 13 800 dol., przy cenie jednego dysku (nośnika) 175 dol. Pierwsze dostawy w Japonii przewidziano na koniec 1983 roku. Przedstawiciel firmy stwierdził, że koszt przechowywania jednego bajtu informacji w tej pamięci stanowi zaledwie 5% kosztu zapamiętywania na dyskach magnetycznych.

Firma 3M rozpoczęła w listopadzie 1982 dostawę dysków optycznych o pojemności 1,6 G bajta. Informacje są zapisywane na dysku o średnicy 305 mm i grubości 4 mm za pomocą lasera półprzewodnikowego o mocy 7—9 mW. W warstwie odpornej na działanie ciepła wypalane są wgłębienia o średnicy 1 μm. Warstwa ta zrobiona jest ze specjalnego, opatentowanego przez firmę materiału. Rozstęp ścieżek wynosi 1,6 μm i tyle samo odstęp pomiędzy poszczególnymi bitami. Dysk obraca się z prędkością 900 obr/min, a moc odczytu wynosi 1 mW.

W 1983 roku przedstawiciele amerykańskich i zachodnioeuropejskich producentów pamięci na dyskach optycznych, a w szczególności firm XEROX, CDC i SHUGART (USA) oraz THOMSON-CSF (Francja) i PHILIPS (Holandia), przyjęli jako standardowe dwa rozmiary zewnętrznej średnicy dysków optycznych — 305 i 356 mm oraz stosunek sygnału do szumu przy odczycie w określonych warunkach — nie mniej niż 45 dB.

W tym samym roku pojawił się nowy dział na rynku optycznych pamięci dyskowych, w postaci videodysków dostosowanych do współpracy z komputerami. Kodowanie informacji cyfrowej w standardowym formacie analogowych dysków wizyjnych pozwala na przechowywanie na nich danych o objętości przekraczającej 1 G bajt. Informację cyfrową na takim dysku może zapisywać tylko jego producent w odróżnieniu od komputerowych dysków optycznych, na których informacje zapisuje bezpośredni użytkownik komputera. Jednakże dyski wizyjne są tańsze i — według oceny specjalizującej się w badaniach rynku firmy LINK RESOURCIES — dyski te stworzą rynek, którego obroty w 1990 roku osiągną poziom 450 mln dol. miesięcznie.

Prace badawczo-rozwojowe w dziedzinie metod przyłączania dysków wizyjnych do komputerów prowadzą firmy LASER DATA, ON-LINE COMPUTER SYSTEMS, ACCESS UNLIMITED (USA) oraz MATROX ELECTRONIC SYSTEMS (Kanada). Dla uzyskania zapisu informacji cyfrowej, firma LASER DATA przekształca zapisane na taśmie magnetycznej „zera" i „jedynki" w sygnały wchodzące na kanał 4,2 MHz, którego parametry są określone formatem ramki wizyjnej. „Jedynki" są tworzone za pomocą modulacji amplitudy sygnału wizyjnego w każdej linii do poziomu czerni lub bieli. Dla zabezpieczenia przed utratą informacji wskutek mikrodefektów dysków, LASER DATA wprowadza dodatkowe bity do każdego słowa informacji. Następnie dane łączone są w bloki po 192 słów, przy czym bity każdego słowa są zawarte w sygnale wizyjnym — za pierwszym bitem pierwszego słowa bloku ma miejsce pierwszy bit drugiego słowa itd. Utrata danych, która mogłaby osiągnąć maksymalnie kilka bitów, zostaje rozdzielona na 192 słowa i poddana procedurze automatycznej korekcji. Przy odczytywaniu danych deszyfrator realizuje przetwarzanie analogowo-cyfrowe i od nowa zbiera bity w 192 słowa wyjściowe. Algorytm korygowania błędów zapewnia porównanie każdego słowa z listą adresów zawartych w programowanej pamięci stałej o pojemności 32 K bajtów. Deszyfrator zapewnia skorygowanie do trzech błędnych bitów.

Jak podał prezes firmy LASER DATA — za kodowanie oryginału dysku wizyjnego 305 mm o pojemności 1,6 G bajta firma będzie pobierać 11 950 dol., natomiast za przygotowanie 100—500 kopii tego dysku — tylko po 55 dol. za każdą kopię.

W 1983 roku wypalanie laserem otworów w dyskach wizyjnych i optycznych oraz dyskach do zapisu sygnałów dźwiękowych odbywało się głównie metodami modulacji wiązki laserowej. Przy zapisie informacji na dyskach wizyjnych, długość impulsów wytwarzanych przez laser modulowana jest amplitudą wizyjnych sygnałów analogowych. Impulsy stosunkowo krótkie odpowiadają sygnałom wizyjnym o małej amplitudzie, natomiast impulsy długie —sygnałom wizyjnym o dużej amplitudzie. Na powierzchni dysku impulsom takim odpowiadają otwory o długości 2—6 μm.

Sposób zapisu sygnałów dźwiękowych jest podobny do zapisu danych na dyskach optycznych. W obu przypadkach wiązka laserowa modulowana jest szeregowym strumieniem danych. Przy zapisie sygnałów dźwiękowych strumień ten składa się ze słów o długości 16 bitów, wytwarzanych przez przetwornik analogowo-cyfrowy, o częstotliwości 44 kHz.

Przy zapisie danych cyfrowych na dyskach optycznych, strumień danych, w zależności od rodzaju przetwarzanych danych, może składać się ze słów 8-, 16- lub 32-bitowych. W obu przypadkach pod działaniem wiązki laserowej w zewnętrznej warstwie dysku wypalane są otwory o średnicy 2 μm lub mniej, przy czym otwory takie interpretują się jako „jedynki", a ich brak — jako „zera". Dyski przeznaczone do zapisu sygnałów dźwiękowych z reguły mają średnicę 102 mm.

Przykładami zastosowania pamięci optycznych na dużą skalę mogą być projekty systemów informatycznych dla Urzędu Patentowego USA oraz Biblioteki Kongresu USA. Projekty są już zatwierdzone i oczekują rychłej realizacji. Przy tworzeniu systemu wyszukiwania informacji patentowej, problemem był sposób optymalnego zapisu 4,5 mln patentów USA i 9 mln patentów zagranicznych. Zdecydowano zapisać teksty tylko 2 mln opisów patentów USA z ostatnich lat, natomiast dla wszystkich 13,5 mln patentów zarejestrować w postaci cyfrowej rysunki i wykresy. Wymagało to pamięci o pojemności 60 miliardów bajtów dla informacji tekstowej oraz ok. 3 trylionów bajtów dla rysunków (objętość informacyjna jednego rysunku ograniczono do 80 K bajtów). Dla rozwiązania tego problemu przyjęto dyski optyczne o łącznej pojemności ponad $3 \cdot 10^{12}$ bajtów. Przy zastosowaniu dysków optycznych o średnicy 305 mm o pojemności 1 G bajta firmy 3M, wymagałoby to ok. 1850 dysków, natomiast w przypadku dysków o pojemności 4 K bajty, jakie w 1984 roku ma produkować firma STORAGE TECHNOLOGY, wystarczy tylko 620 dysków.

Według ostatnich informacji seryjna produkcja podsystemu pamięci optycznej typ 7600 firmy STORAGE TECHNOLOGY rozpocznie się w drugim kwartale 1984 roku. Firma gwarantuje trwałość zapisu na dysku w ciągu dziesięciu lat. System wykrywania błędów zapewnia wysoki poziom pewności zapisu, wyrażający się prawdopodobieństwem pojawienia się jednej pomyłki na 10 trylionów bitów. Dyski o średnicy 356 mm są przechowywane w ochronnej kasecie, automatycznie usuwanej przy wprowadzaniu napędu pamięci. Szybkość zapisu i odczytu wynosi 3 mln znaków/s. Jednostka pamięci składa się z napędu typu 7640, który kosztować będzie ok. 130 tys. dol. i z bloku sterowania typu 8880 w cenie ok. 65 tys. dol. Cena jednego dysku, w zależności od wielkości dostawy, wynosić będzie 140—225 dol. Pamięć wyposażona jest w zabezpieczenie programowe, za które pobierana jest jednorazowa opłata 1000 dol., a następnie po 400 dol. miesięcznie, przy czym firma zapewnia stałą modyfikację oprogramowania oddanego do dyspozycji użytkownika.

Dla zapisu informacji o zbiorach Biblioteki Kongresu USA, przewidziano wykorzystanie dysków optycznych produkcji francuskiej firmy THOMSON-CSF. System o nazwie „Image Link" o koszcie ok. 1,7 mln dol. umożliwia sprowadzenie do postaci cyfrowej i zapisanie na dysku, kosztem zaledwie 2 centów, informacji zawartej na arkuszu o wymiarach 216 × 280 mm, przy rozdzielczości 120 × 120 elementów obrazu na 1 cm². Odpowiada to szybkości przetwarzania 8 mln elementów obrazu/s. Po zapisie informacji, dyski przeszukiwane są za pomocą wiązki laserowej, a ich zawartość jest wyświetlana na ekranie monitora o dużej zdolności rozdzielczej (2456 × × 1644 linii). Informacje tekstowe będą wyświetlane na ekranie o rozdzielczości 60 × 120 elementów na 1 cm². Tak duża rozdzielczość jest niezbędna ze względu na to, że w dokumentach często występuje jasny druk, a wiele tekstów z upływem czasu blednie i zaciera się.

W systemie tym przewidziano dyski średnicy 305 mm o pojemności 1 G bajt. System, zapewniający automatyczny wybór i odtwarzanie informacji z jednego ze stu wykorzystywanych dysków optycznych, pozwala przechowywać dane dotyczące ok. 1 mln książek. Dostawy systemu „Image Link" miały rozpocząć się w październiku 1983, przy czym jego cena — według oświadczenia prezesa firmy INTEGRATED AUTOMATION (USA), instalującej ten system — jest około dwukrotnie niższa od ceny systemu mikrofilmowego o analogicznych parametrach.

Dysk optyczny firmy THOMSON-CSF ma strukturę dwuwarstwową: warstwę polimeru i metaliczny nośnik zapisu umieszczony w warstwie z tworzywa sztucznego. Materiał nośnika zapisu ma, według opinii producenta, znacznie lepszą odporność na wymazywanie, utlenianie i wpływ wilgoci w porównaniu do stosowanego dotąd stopu telluru. Tworzywo sztuczne służy zarówno jako powłoka ochronna, jak i podłoże dla nośnika zapisu. Według wypowiedzi dyrektora wydziału dysków optycznych firmy THOMSON-CSF — już w 1988 roku cena dysków o pojemności 1 G bajta spadnie do ok. 50 dol., a jednostka pamięci kosztować będzie nie więcej niż 5 tys. dol.

Aby zapisać informację na dysk, warstwę metaliczną nagrzewa się wiązką lasera półprzewodnikowego o mocy 15 mW. Pod wpływem nagrzewania następuje lokalny rozkład polimeru, przy czym podwyższa się ciśnienie gazu, tak że w warstwie metalicznej tworzą się wybrzuszenia rzędu 0,8 μm. Takie wybrzuszenia ujawniają się przy odtwarzaniu punktową wiązką lasera o mocy 1 mW, w czasie gdy dysk obraca się z prędkością 1200 obr/min. Firma THOMSON-CSF prowadzi obecnie prace badawcze nad możliwością zapisu danych na obu powierzchniach dysku.

Firma SHUGART opracowała pamięć na dyskach optycznych typu „Optimen 1000" o pojemności 1 G bajt. Jednostka pamięci wykorzystująca wymienne dyski o średnicy 305 mm sprzedawana jest w cenie 6 tys. dol. Wbudowane systemy ochrony zapisu zapewniają nienaruszalność zapisanych informacji. Zapis danych odbywa się za pomocą lasera półprzewodnikowego z gęstością $2 \cdot 10^8$ bitów/cal² przy liczbie 14,5 tys. ścieżek na cal, co przewyższa wartość tego parametru spotykanego w standardowych pamięciach na dyskach magnetycznych. Liniowa gęstość zapisu wynosi 15 tys. bitów/cal. Dysk dostarczany jest użytkownikowi z naniesionymi przez wytwórcę ścieżkami synchronizacji oraz kodami umożliwiającymi dokładne rozmieszczenie ścieżek i ustawienie głowic. Średni czas dostępu do informacji zapisanej na dysku wynosi 100 ms, przy czym dane grupuje się sektorami po 1024 bajty. Pamięć jest wyposażona we wbudowany blok diagnostyki uszkodzeń i zapewnia możliwość przyłączenia sprzęgów, przeznaczonych do wykorzystywania tradycyjnych pamięci na dyskach elastycznych lub typu Winchester. Do jednego urządzenia o wymiarach 175 × 475 × 600 mm, ciężarze 22 kg i mocy pobieranej 230 W można dołączyć do ośmiu jednostek dyskowych.

Dysk optyczny w urządzeniu „Opimen" ma strukturę trzywarstwową, przy czym warstwa zewnętrzna stanowi ochronę zabezpieczającą przed zanieczyszczeniem warstwy zawierającej zapis informacji. Wiązki laserowe — zapisująca i odczytująca — przechodzą przez dolną warstwę ochronną o grubości 1 mm, przepuszczającą światło, przy czym warstwa ta spełnia również rolę podłoża. Głowice optyczne zapisu-odczytu umieszczone są w odległości 1—2 mm od powierzchni dysku. Jest to odległość kilkaset razy większa od stosowanej w pamięci typu Winchester. Pozwala to wykorzystywać stosunkowo tanie dyski wymienne — bez potrzeby dokładnej filtracji powietrza i stosowania drogiej kasety wymiennej, niezbędnej w pamięciach Winchester.

Pamięć opracowana przez japońską firmę MATSUSHITA ELECTRIC INDUSTRIAL wykorzystuje dyski optyczne o średnicy 203 mm i pojemności 700 M bajtów z liniową gęstością zapisu 15 tys. bitów/cal oraz gęstością poprzeczną 6500 ścieżek/cal. Średni czas wybrania informacji wynosi 500 ms, a szybkość wymiany informacji — 5 M bitów/s. Przy większych dostawach cena tej pamięci wynosi ok. 35 tys. dol. (50 dol. za 1 M bajt).

W połowie listopada 1983 firma NCR poinformowała o planowanej na koniec 1984 roku produkcji pamięci na dyskach optycznych, przeznaczonej dla komputerów osobistych. Cena tych pamięci o pojemności 100 M bajtów nie powinna przekroczyć 1000 dol. Pojemność pamięci powinna zapewnić wyświetlenie na ekranie monitora co najmniej 50 tys. kadrów. Firma uprzedziła, że do połowy lat osiemdziesiątych nie należy oczekiwać pojawienia się pamięci na dyskach optycznych z możliwością powtórnego zapisu, przeznaczonych dla tanich komputerów.

Pamięć opracowana przez firmę PHILIPS wykorzystuje dwustronne dyski optyczne o średnicy 305 mm i pojemności 2500 M bajtów, z gęstością liniową zapisu 15 tys. bitów/cal i gęstością poprzeczną 8 tys. ścieżek/cal oraz średnim czasem wybrania informacji 500 ms i szybkością wymiany 5—10 M bitów/s. Przy większych dostawach cena pamięci odpowiada wskaźnikowi 10 dol. za 1 M bajt.

Amerykański oddział firmy HITACHI rozpoczął dostawę pamięci typu H-6975I z wykorzystaniem dysków optycznych średnicy 305 mm o pojemności 1310 M bajtów i średnim czasie dostępu do informacji 250 ms. Firma opracowuje obecnie pamięć typu H-6975-A1 z 16 dyskami o średnim czasie dostępu 5 s, a także podsystem typu H-6975-A2 z 32 dyskami o średnim czasie dostępu 6 s. Realizowany jest także system „Hitfile" do przechowywania i wyszukiwania danych. System ten obejmuje do ośmiu terminali, za pomocą których można zmieniać skalę odczytywanych dokumentów. Miesięczny koszt dzierżawy jednego dysku optycznego wynosi ok. 140 funtów, a minimalnej konfiguracji systemu „Hitfile" — ok. 1000 funtów.

Niezależnie od wielkiego zainteresowania dyskowymi pamięciami optycznymi dla komputerów, należy stwierdzić, że proces ich opracowywania jest bardzo trudny. Wiadomo już, że np. firma BURROUGHS wstrzymała w 1983 roku prace nad opracowaniem pamięci i dysków optycznych o dużej pojemności. Pierwotnie firma ta planowała w połowie 1984 rozpocząć dostawę pamięci dyskowych o pojemności 2—4 G bajtów i czasie wybierania 50—100 m/s. Po specjalnych badaniach kierownictwo firmy nie zaakceptowało programu uruchomienia produkcji tych pamięci. Po zwolnieniu 90 inżynierów zatrudnionych w pracach badawczych, jedynie 10—15 osób będzie kontynuować podstawowe prace w tym temacie. Jednakże większość zwolnionych specjalistów kontynuuje analogiczne prace w innych firmach.

Firma BUSINESS COMMUNICATIONS (USA) przeprowadziła analizę rynku pamięci o dużej pojemności i tendencji jego rozwoju. Analiza ta wykazała, że w 1983 roku sprzedano pamięci o dużej pojemności różnych typów za sumę ok. 8,8 mld dol., przy czym do 1988 roku średnie tempo rozwoju tego rynku wynosić będzie 23% rocznie. Udział pamięci na dyskach magnetycznych stanowi obecnie ok. 40%. Najszybszy rozwój (70% rocznie) powinny wykazać dyskowe pamięci optyczne. Obecnie są prowadzone intensywne prace nad zbudowaniem pamięci na dyskach optycznych z możliwością wymazywania i ponownego zapisu informacji.

Japońska firma SHARP wykorzystała stop terb-dysproz-żelazo do wytworzenia ciemnego, niekrystalicznego materiału, który nakłada się na wykonane z miedzi podłoże, odbijające światło. Przy nagrzaniu wiązką laserową, na dysku tworzą się obszary odbijające światło od warstwy miedzi znajdującej się pod warstwą pokrywającą. Firma podaje, że na dyskach tych można dokonywać do miliona operacji zapisu.

Również japońska firma PANASONIC podała o wykorzystaniu do wyrobu dysków optycznych z wielokrotnym zapisem pokrycia wykonanego z podtlenku telluru z dodatkiem niewielkich ilości germanu, indu i ołowiu. Pokrycie to w normalnych warunkach jest amorficzne, ale pod działaniem wiązki laserowej jego określone obszary przyjmują strukturę krystaliczną o właściwościach odbijających, co odpowiada „jedynce", natomiast „zeru" — obszary amorficzne, nie odbijające wiązki laserowej. Pod działaniem wiązki wymazującej cała powierzchnia dysku odzyskuje ponownie właściwości materiału amorficznego. Do zapisu i odczytu wykorzystuje się tu laser GaAs o długości fali 830 nm, przy czym potrzebna do zapisu moc wynosi 8 mW, a do odczytu — 1 mW. Do wymazywania zapisu wykorzystuje się laser o długości fali 780 nm i mocy 10 mW, przy czym wymazywanie odbywa się w obszarach 1 × 10 µm. Prędkość dysku wynosi 1800 obr./min, a więc jest dwukrotnie większa niż w urządzeniach z dyskami niewymazywalnymi, jakie dostarczano w 1983 roku i przygotowywano do sprzedaży w 1984.

Przewiduje się, że seryjna produkcja pamięci z dyskami wymazywalnymi rozpocznie się dopiero w roku 1986. Ceny tych urządzeń będą o 40—50% większe od cen urządzeń z dyskami niewymazywalnymi.

Bardzo obiecującymi wydają się być pamięci magnetyczno-optyczne z dyskami wymiennymi. W pamięciach tych materiały amorficzne nagrzewają się przez oświetlenie wiązką laserową w obecności silnego pola magnetycznego, przy czym kierunek tego pola związany jest z algorytmem działania lasera.

Prototypowa pamięć na dysku o średnicy 51 mm o pojemności ok. 10 M bajtów (co odpowiada ok. 5 tys. stron tekstu) była demonstrowana przez firmę PHILIPS RESEARCH LAB. Do zapisu „jedynki": amorficzna warstwa metali ziem rzadkich — gadolinu i stopu terb-żelazo — nagrzewa się do 100°C za pomocą lasera AlGaAs o mocy 3 mW. Wiązka laserowa sterowana jest cewką magnetyczną na powierzchni dysku, na której regeneruje się owalny obszar o wymiarach 5 × × 2 µm. Czas zapisu jednego bitu wynosi 3 µs. System magnetooptyczny zapewnia czas dostępu rzędu 50 ms i szybkość wymiany informacji 250 K bitów/s. Chociaż seryjna produkcja podobnych urządzeń jest spodziewana nie wcześniej niż w 1985 roku, badania prototypu pozwoliły już rozstrzygnąć istotną wątpliwość, wykazując, że magnetooptyczne pamięci dyskowe mogą zapewnić uzyskanie potrzebnego stosunku sygnału do zakłócenia.

Przewiduje się, że w przyszłości znacznie wzrosną różnice pomiędzy możliwościami pamięci na dyskach optycznych i pamięci na dyskach magnetycznych. Oczekuje się, że w 1990 roku gęstość zapisu na dyskach magnetycznych osiągnie $5 \cdot 10^7$ bitów/cal²,

a na dyskach optycznych 10^9 bitów/cal^2. W 2000 roku analogiczne wskaźniki wynosić będą odpowiednio $8 \cdot 10^7$ bitów/cal^2 i $6 \cdot 10^9$ bitów/cal^2 Bliższym w czasie udoskonaleniem pamięci na dyskach optycznych będzie niewątpliwie możliwość przechowywania na obu powierzchniach dysku oraz zwiększenie pojemności poszczególnych powierzchni do 4 G bajtów w wyniku udoskonalenia techniki kodowania zapisu informacji.

Kontynuowany jest również proces opracowywania udoskonalonych laserów do zapisu na dyskach optycznych.

We wrześniu 1983 japońska firma HITACHI uruchomiła doświadczalną produkcję lasera półprzewodnikowego do zapisu informacji na dyskach optycznych. Produkcja seryjna tego lasera miała rozpocząć się w marcu 1984. Działa on w zakresie światła widzialnego na fali długości 0,78 μm z mocą wyjściową 40 mW. Ten wysoki poziom mocy wyjściowej osiągnięto przez zastosowanie pokrycia wielowarstwowego o dużym współczynniku odbicia, które nanosi się na jedną ze ścianek rezonatora laserowego. Pokrycie to składa się z kolejnych warstw dwutlenku krzemu i krzemu amorficznego, których grubość odpowiada połowie roboczej długości fali lasera. Dzięki takiemu pokryciu współczynnik odbicia wzrasta z 30 do 90%. W rozwiązaniu tym na przeciwną ściankę rezonatora skierowane jest 95% energii świetlnej, wytwarzanej w laserze. Firma HITACHI zapewnia, że przyjęcie takich pokryć w istniejących laserach pozwala na zwiększenie poziomu mocy wyjściowej o 50%. Zwiększenie tego poziomu ułatwiane jest przez płaski światłowód o specjalnej konstrukcji.

Opracował

JAN RYŻKO

na podstawie „Ekspress Informacji TC-2 Sredstwa wyczislitielnoj tiechniki i orgtechniki", wypusk 2, Moskwa, 1984

Najeźdzcy ze wschodu

Nie tylko my — Polacy, mamy wschodnie fobie. Ma je również — na swój sposób — świat zachodni. Powszechnie wiadomo, że Japończycy są bardzo silnym konkurentem w zakresie wielu produktów konsumpcyjnych. Po japońskich aparatach fotograficznych, pojawił się sprzęt HiFi i VIDEO, potem motocykle i wreszcie samochody. Teraz przyszła kolej na mikrokomputery.

Cóż takiego wymyślili Japończycy, co mogłoby zachwiać pozycją firm ACORN czy SINCLAIR, że prasa brytyjska zaczęła bić na alarm? Dlaczego pospiesznie zaczęto krytykować zasadę „dziel i rządź", bezkarnie przynoszącą do tej pory zyski potentatom w produkcji komputerów powszechnego użytku?

Pomysł Japończynków jest w sumie banalny — w chwili gdy popyt na mikrokomputery do użytku domowego jest już na tyle rozbudzony, że nie trzeba ludzi przekonywać do nabycia mikrokomputera — jedynym atutem może być duża ilość oprogramowania. W sytuacji gdy poszczególne typy mikrokomputerów w sposób istotny różnią się od siebie, tak jak to ma miejsce obecnie, produkcja oprogramowania jest szalenie rozproszona. Los, jak wiadomo, jest złośliwy i zawsze ten najciekawszy i najbardziej użyteczny program oferowany jest dla innego komputera niż posiadany. Programowa kompatybilność mikrokomputerów wielu różnych firm najprawdopodobniej przesądzi o sukcesie zaproponowanego przez Japończyków standardu MSX.

Tak, jak w swoim czasie na sukces firmy APPLE pracował cały świat, wymyślając oprogramowanie i dodatkowe pakiety sprzętowe, tak należy się spodziewać, że niebawem większość firm produkujących oprogramowanie skupi się na ofertach dla mikrokomputerów z rodziny MSX. To się po prostu zacznie opłacać. Ponad tuzin japońskich firm — wedle początkowych zamierzeń — będzie eksportować na rynek zachodni ponad milion mikrokomputerów rocznie.

Podstawą standaryzacji oprogramowania komputerów MSX jest wersja języka BASIC opracowana przez firmę MICROSOFT. Na tym jednak nie koniec. Wiadomo, że na obecnym etapie rozwoju techniki mikrokomputerowej znaczna część oprogramowania pisana jest z wykorzystaniem języka maszynowego (ASSEMBLER). Otóż wszystkie komputery MSX będą wykorzystywały ten sam mikroprocesor Z80. Idąc dalej — we wszystkich będzie zastosowany ten sam układ sterownika CRT. Grafika w 16 kolorach z rozdzielczością 256 na 192 punkty stwarza duże możliwości, wymaga jednak dobrej obsługi programowej. Dla wszystkich mikrokomputerów MSX będzie można stosować te same podprogramy sterujące pracą układu TMS 9918A! Wybranie tego właśnie układu w pewien sposób określa odbiorcę, do którego adresowane są komputery z rodziny MSX. Możliwość wykorzystania 32 „duszków" (ang. sprite) daje do ręki poważne narzędzie producentom gier komputerowych. Za grami zwykle podąża oprogramowanie do celów edukacyjnych. A więc znakomity komputer do wykorzystania w domu lub w szkole, ale nie w biurze, gdzie 32 litery w linijce tekstu to stanowczo za mało.

Trzy kanały dźwiękowe o rozpiętości ośmiu oktaw (układ AY-3-8910) to dużo dla tzw. masowego odbiorcy, lecz z pewnością nie wystarczy muzykom profesjonalnym.

Parametry video i muzyczne sugerują więc, że mikrokomputery MSX nie zamierzają konkurować ze sprzętem profesjonalnym typu IBM PC, LISA czy OLIVETTI M20. Zaskoczyć może więc fakt, że wszystkie MSX-y mają mieć możliwość zaadresowania aż 1 MB pamięci. I faktycznie jest to raczej perspektywa na przyszłość, gdyż obecne konstrukcje zawierają od 16 KB do 96 KB.

Warto jeszcze wspomnieć, że standard opisuje także sprzęg z urządzeniami zewnętrznymi: manetką do sterowania (ang. joystick), kasetami na dodatkową pamięć (w tym stałą, zawierającą oprogramowanie) i urządzeniami rozszerzającymi możliwości podstawowego modułu komputera.

Komputery z rodziny MSX są już produkowane w Japonii w liczbie ponad 50 tys. egzemplarzy miesięcznie. Jako pierwsza wystartowała firma SANYO (w zaledwie trzy miesięce po ogłoszeniu standardu). Mikrokomputer MPC5, przypominający z wyglądu TI 99/4 firmy TEXAS INSTRUMENTS, ma „prawdziwą" klawiaturę, 32 KB pamięci RAM i dwa gniazda na przystawki. Kosztuje on ok. 130 funtów [1]). MPC 10 został dodatkowo wyposażony w pióro świetlne, które może służyć do wyboru programowych opcji spośród zestawu wyświetlanego na ekranie. Kosztuje 150 funtów. Najdroższy komputer firmy SANYO o nazwie MPC-X kosztuje 190 funtów i został wyposażony w dodatkowy tryb graficzny o rozdzielczości 500 na 200 punktów. Stwarza to już możliwości zastosowań profesjonalnych — tym bardziej, że firma zapowiada możliwość rejestrowania obrazów telewizyjnych.

[1]) Podawane ceny dotyczą Japonii, aby uniknąć zniekształceń cytuję je w funtach angielskich tak, jak były podawane w tekstach źródłowych

Firma JVC proponuje mikrokomputer o nazwie HC-5, również o rozbudowanych możliwościach graficznych, jednak z tanią klawiaturą i z niewielką pamięcią RAM — 16 KB. HC-5 kosztuje 150 funtów.

Nieco rozczarowuje propozycja firmy SONY, znanej z umiejętności łączenia tanich rozwiązań z dobrą jakością. Mikrokomputer Hit-Bit HB-55 ma prymitywną klawiaturę i 16 KB pamięci RAM [2]) (z czego użytkownik może wykorzystać 12,5 KB). Interesująca jest jednak zaoferowana przez SONY przystawka pamięci. Zawiera ona 4 KB pamięci RAM, wykonanej w technologii CMOS i zasilanej z baterii. Dane nie są więc tracone po odłączeniu zasilania!

Firma HITACHI proponuje za 150 funtów komputer MBH-1 zawierający 96 KB pamięci RAM.

Mikrokomputer CANON B-10 również kosztuje 150 funtów, lecz z wyglądu przypomina komputery przeznaczone do zastosowań biurowych. Jak przewiduje standard, ma on 5 „programowalnych" klawiszy, ale jak w przypadku większości komputerów z rodziny MSX — poprzez użycie klawisza SHIFT można przypisać im dziesięć funkcji.

Z kolei FM-X firmy FUJITSU, mimo że wygląda jak łachmaniarz, jest wyposażony w niemal „żywą" klawiaturę — tyle bowiem różnych funkcji można przypisywać poszczególnym klawiszom.

Firma MITSUBISHI, znana z produkcji gigantycznych statków i samochodów, a także robotów, proponuje do swojego mikrokomputera ML-8000 z pamięcią 32 KB przystawkę w postaci manipulatora. Zapowiadane są dalsze roboto-podobne urządzenia mogące współpracować z komputerami MSX.

CF-2000 to propozycja firmy NATIONAL obejmująca wersję z 64 i 16 KB pamięci RAM. Można też nabyć kolorową drukarkę — ploter (format A4) za niecałe 200 funtów.

Firmy, które zgłosiły akces do MSX:

CANON	NEC
FUJITSU	PHILIPS
GENERAL	PIONEER
HITACHI	SANYO
JVC	SONY
KYOCERA	SPECTRAVIDEO
MATSHUSHITA	TOSHIBA
MITSUBISHI	YAMAHA

Najbardziej oryginalna jest chyba propozycja firmy JAMAHA. Do komputerów YIS-505 i XC-5 można przyłączyć klawiaturę organów elektronicznych i syntetyzator. XC-5 może ponadto sterować całym zestawem syntetyzatorów. Również firma PIONEER proponuje połączenie komputera ze sprzętem muzycznym, jednak przeznaczonym do odtwarzania. Komputer będzie można wmontować w wieżę ze sprzętem HiFi i wykorzystać do sterowania całym zestawem.

Na tym jednak nie koniec rewelacji związanych z MSX-em. Firma MICROSOFT, która opracowała podstawowe oprogramowanie, jak również ogłosiła sprzętowe wymagania standardu, przygotowała projekt układu scalonego, który w jednym kawałku krzemu będzie realizował wszystkie funkcje sprzętowe wymagane przez standard MSX. W chwili gdy tekst ten dotrze do Czytelników, pozostaną już tylko pojedyncze miesiące do rozpoczęcia produkcji tego układu. Według zapowiedzi może on zredukować cenę mikrokomputerów z serii MSX do ok. 30 funtów! W związku z tym producenci wszelkiego sprzętu powszechnego użytku zaczęli deklarować chęć wbudowania komputerów MSX w swoje wyroby. Prawdę mówiąc — trudno raczej wyobrazić sobie zastosowania lodówki z serii MSX.

Oczywiście są i głosy sceptyczne w sprawie standardu MSX. Ponieważ producenci ze względów konkurencyjnych będą wyposażali swoje wyroby w różne rozszerzenia, mikrokomputery MSX ponownie przestaną być kompatybilne. Równie często wysuwany jest argument, że ujednolicenie sprzętowe spowoduje nagminne nieautoryzowane (tzn. nie przynoszące wpływów producentom) kopiowanie oprogramowania.

Sukces standardu MSX wydaje się jednak nieuchronny. Stąd i zainteresowanie firm spoza Japonii. Firma SPECTRAVIDEO (USA, Hong Kong) jako pierwsza weszła na rynek europejski z komputerem z rodziny MSX. Podobno PHILIPS już podpisał akces, a rozmowy dwóch dużych firm brytyjskich są w toku. Jak twierdzą autorzy standardu, MSX jest otwarty dla tych wszystkich, którzy zechcą się do niego zastosować i uiścić opłatę „wpisową".

Na zakończenie warto wspomnieć, że układy wykorzystywane w konstrukcji mikrokomputerów MSX są stosunkowo łatwo dostępne na Zachodzie i niezbyt drogie (Z80A — 4 dol., AY-3-8910 — 10 dol., TMS 9918A — 40 dol.). Porównując mikrokomputery oferowane dotychczas przez firmy „polonijne", nie powinno być większych problemów z wykonaniem rodzimej wersji MSX.

Kto pierwszy otworzy nam okno na świat?

ANDRZEJ J. PIOTROWSKI

LITERATURA

[1] MSX Invaders. Your Computer, March 1984

[2] MSX — What is it? Personal Computer World, March 1984.

Kalendarz 1985

W kalendarzu zapowiadamy tylko te imprezy, o których zostaliśmy poinformowani bezpośrednio przez organizatorów. Podajemy tylko główne informacje — szczegółowe otrzymać można w redakcji.

WIOSNA

- HARDWARE AND SOFTWARE COMPONENTS AND ARCHITECTURES FOR THE 5th GENERATION — Paryż, 5—7 marca, organizator: AFCET INFORMATIQUE
- MIKRONIKA-85, Postęp w budowie precyzyjnego sprzętu elektroniczno-mechanicznego — Warszawa, 15—16 maja, organizatorzy: SIMP, Politechnika Warszawska
- SM 85 — Warszawa, 21—24 maja, organizator: Klub Użytkowników Minikomputerów SM, Polskie Towarzystwo Informatyczne
- PROLAMAT 1985, Software for Discrete Manufacturing — Paryż, 11—13 czerwca, organizator: AFCET

LATO

- IX KRAJOWA KONFERENCJA AUTOMATYKI, Stan i perspektywy automatyki w Polsce — Łódź, koniec czerwca, organizatorzy: Polski Komitet Pomiarów i Automatyki NOT, Politechnika Łódzka
- SIGGRAPH 85, The Twelft Annual Conference on Computer Graphics and Interactive Techniques — San Francisco, 22—26 lipca, organizator: ACM
- WCCE-85, World Conference on Computers in Education — Norfolk, Virginia, USA, 29 lipca—2 sierpnia, organizatorzy: IFIP/TC-3, AFIPS
- EUROMICRO 85, Eleventh Symposium on Microprocessing and Microprogramming; Microcomputers, usage and design — Bruksela, 3—6 września, organizator: EUROMICRO
- ZASTOSOWANIE KOMPUTERÓW W PRZEMYŚLE, IV konferencja naukowo-techniczna — Szczecin, 11—13 września, organizatorzy: OW NOT, Politechnika Szczecińska

JESIEŃ

- VIII MIĘDZYNARODOWE SEMINARIUM NT. SYSTEMÓW ZARZĄDZANIA BAZĄ DANYCH — Pieštany (Czechosłowacja), 30 września — 4 października, organizator: VUSEI-AR, Bratysława

Wilczyński W.: PLZ — język programowania mikroprocesorów (1)

INFORMATYKA 1984, nr 12, s. 1

Pierwsza część charakterystyki języka wysokiego poziomu PLZ, przeznaczonego do programowania mikroprocesorów. Omówiono struktury danych oraz podstawowe operacje, ilustrowane licznymi przykładami.

Kott R., Szwed K.: Uniwersalne asemblery dla mikrokomputerów

INFORMATYKA 1984, nr 12, s. 4

Przegląd zrealizowanych uniwersalnych asemblerów dla mikrokomputerów oraz charakterystyka rozwiązań asemblera uniwersalnego UAS, zrealizowanego przez autorów na Politechnice Warszawskiej.

Rybus R.: Metody realizacji programów uruchomieniowych dla mikroprocesorów

INFORMATYKA 1984, nr 12, s. 8

Charakterystyka podstawowych metod realizacji programów uruchomieniowych dla mikroprocesorów: pułapek programowych oraz krokowego wykonywania programu. Podano kryteria oceny i dokonano porównania tych metod.

Zieleniewski R.: Problemy programowania mikroprocesorów

INFORMATYKA 1984, nr 12, s. 19

Omówienie czynników powodujących trudności programowania mikroprocesorów oraz koncepcji języka pozwalającego zmniejszyć te trudności.

Вильчиньски В.: PLZ — язык программирования микропроцессоров (1)

INFORMATYKA 1984, № 12, стр. 1

Первая часть характеристики языка высокого уравня PLZ, предназначенного для программирования микропроцессоров. Обсуждение структуры данных и основные операции, иллюстрированные многими примерами.

Котт Р., Швед К.: Универсальные ассемблеры для микро-ЭВМ

INFORMATYKA 1984, № 12, стр. 4

Обзор существующих универсальных ассемблеров для микро-ЭВМ и характеристика решений универсального ассемблера UAS, яадработанного авторами в Варшавском политехническом институте.

Рыбус Р.: Мотоды реализации программ отладки для микропроцессоров.

INFORMATYKA 1984, № 12, стр. 8

Характеристика основных методов реализации программ отладки для микропроцессоров: программных ловушек и шагового выполнения программы. Даны критерии оценки и сравнение этих методов.

Зеленевски Р.: Проблемы програмирования микропроцессоров.

INFORMATYKA 1984, № 12, стр. 19

Обсуждение факторов вызывающих трудности программирования микропроцессоров и концепции языка позволяющего уменьшить эти трудности.

Wilczyński W.: PLZ — a programming language for microprocessors (1)

INFORMATYKA 1984, No. 12, p. 1

First part of characteristics of the PLZ, a high level language for microprocessor programming. Data structures and basic operations, illustrated with many examples, are discussed.

Kott R., Szwed K.: Universal assemblers for microcomputers

INFORMATYKA 1984, No. 12, p. 4

Survey of realized universal assemblers for microcomputers and characteristics of UAS universal assembler solution, realized by the authors in the Warsaw Technical University.

Rybus R.: Realization methods for debugging microprocessor software

INFORMATYKA 1984, No. 12, p. 8

Characteristics of basic methods for realization of debugging microprocessor software: traps and single-step program execution. Evaluation criteria and comparison of methods are presented.

Zieleniewski R.: Problems of microprocessor programming

INFORMATYKA 1984, No. 12, p. 19

Discussion of difficulty factors in microprocessor programming and conception of the language, which decreases such difficulty.

Wilczyński W.: PLZ — Eine Programmiersprache für Mikroprozessoren (1)

INFORMATYKA 1984, Nr. 12, S. 1

Erster Teil einer Charakteristik von höherere Sprache PLZ, die für Programmierung von Mikroprozessoren bestimmt ist. Es wurden Datenstrukturen und Grundoperationen, mit zahlreichen Beispielen illustriert, besprochen

Kott R., Szwed K.: Universelle Assembler für Mikrorechner

INFORMATYKA 1984, Nr. 12, S. 4

Ein Übersicht von realisierten universellen Assemblern für Mikrorechner und eine Charakteristik der Lösungen von UAS — Universalassembler, der von den Autoren in der Warschauer Technischen Universität realisiert wurde.

Rybus R.: Methoden zur Ausführung der Fehlersuchprogramme für Mikroprozessoren

INFORMATYKA 1984, Nr. 12, S. 8

Eine Charakteristik der Grundmethoden zur Ausführung der Fehlersuchprogramme für Mikroprozessoren: Programmfällen und schrittweisen Programmausführung. Es wurden Beurteilungskriterien und ein Vergleich dieser Methoden angegeben.

Zieleniewski R.: Probleme der Mikroprozessorprogrammierung

INFORMATYKA 1984, Nr. 12, S. 19

Eine Besprechung der Faktoren, die Schwierigkeiten der Mikroprozessorprogrammierung verursachen, und des Entwurfes einer Sprache, die diese Schwierigkeiten vermindert.

Czy nasze poglądy są kompatybilne?

Nawet gdybym nie czytał INFORMATYKI musiałbym zareagować na ośmiokrotne użycie mojego nazwiska w tekście o objętości niecałej kolumny (p. Władysław Klepacz, Konieczność kompromisu, INFORMATYKA, nr 7, 1984). Autor przedstawia w nim swój pogląd „na temat kształtowania prawidłowych terminów informatycznych". Sprowadza się on do tego, że w przypadku tworzenia nowych terminów należy zachować względną tolerancję w stosunku do oryginalnych nazw pochodzenia angielskiego.

Ogólnie, duża część sporów terminologicznych dotyczących słownictwa informatycznego wiąże się z problemem przyswajalności terminów anglojęzycznych. Podstawowe pytanie, które można w związku z tym postawić, brzmi: czy można ją przyswajać bezkrytycznie. Z nieoczekiwaną pomocą przychodzi mi dziedzina bardzo odległa, ale napotykająca na te same trudności terminologiczne.

W stałej rubryce niedzielnego wydania „Życia Warszawy' — „Byki i byczki" — Ibis tak wyraża swój stosunek do obfitości słownictwa anglojęzycznego w muzyce młodzieżowej: „Nie jestem przeciwny przejmowaniu z innych języków zwrotów i wyrażeń, dla których nie mamy polskich nazw (...) Powtarzam: nie jestem przeciwnikiem zapożyczeń, bo przecież nie żyjemy na bezludnej wyspie (...). Jestem jednak przeciwny wprowadzaniu wyrazów obcych wszędzie tam, gdzie mamy własne terminy, określające w miarę dokładnie to, o co chodzi".

Zgoda. Historia zapożyczeń jest co najmniej tak długa, jak historia narodu. Warto jednakże zastanowić się jak wprowadzać terminy obcojęzyczne, gdy jest to uzasadnione brakiem odpowiedników rodzimych. Przy zapożyczaniu terminów też powinny obowiązywać pewne zasady.

Takie terminy, jak **adres**, **akumulator**, **program**, **rejestr**, **translator**, a od niedawna — **edytor** i **port**, to wyrazy obce wprowadzone do języka informatycznego bezboleśnie, gdyż istniały już w języku polskim wcześniej, a w informatyce nadano im tylko nowe znaczenie.

Inne — jak **bit**, **kod**, **modem**, **multiplekser**, **terminal**, wcześniej **algorytm** i **komputer**, a ostatnio **asembler**, **emulator** i **kursor**, przeniesiono wprost z języka angielskiego, wprowadzając je jako nowe wyrazy do języka polskiego, gdyż nie pozostawały w sprzeczności z rodzimymi zasadami językowymi. W tej sytuacji neologizmy rdzennie polskie zapewne zostałyby wyparte.

Tu jednak dochodzimy do granicy tolerowania zapożyczeń. Czy powszechnie dziś używany wyraz **bajt** (ang. **byte**) został poprawnie utworzony? Mam poważne obawy, że nie. Dążenie do pisowni wyrazu zgodnej z wymową jest chwalebne, bo taka sytuacja ułatwia życie. Co jednak robić, jeżeli wymowa obcego wyrazu odbiega od jego pisowni? W książce „Kryteria poprawności językowej" prof. Witold Doroszewski pisał: „Jako wytyczną przyjąć należy, że we wszystkich tekstach pisanych zachowuje się obce nazwy własne w ich oryginalnej pisowni". Choć trudno uznać tę zasadę za nienaruszalną w stosunku do wszystkich, także pospolitych wyrazów, pochodzenia obcego (mam świadomość, że ostatecznej weryfikacji terminów dokonuje życie), to głosuję stanowczo przeciwko jej całkowitemu ignorowaniu. Zresztą w informatyce bywa ona również przestrzegana — przecież nie piszemy ani nie mówimy **kompiuter**, lecz **komputer**; nie piszemy, choć mówimy **onlajn** i **oflajn** (ang. on-line, off-line).

To jest przyczyna, dla której na oznaczenie sprzęgu można dopuścić wyraz **interface** (w oryginalnym brzmieniu), ale należy zwalczać — **interfejs**, póki na odrzucenie go nie będzie za późno, tak jak w przypadku wyrazu **bajt**. Ale nie jest to przyczyna jedyna. Dopuszczając fonetyczną pisownię terminów obcojęzycznych, automatycznie zezwalamy nie tylko na posługiwanie się wyrazem **interfejs**, ale na używanie wyrazów takich jak **teletajp**, **displej**, **tajmer** itp. (ang. teletype, display, timer). Takie są skutki uznania spolszczonej pisowni wyrazu **interface** przez Polską Normę.

Nie trzeba chyba dalej uzasadniać, że ten sposób zapożyczania jest dość wulgarny, bo nie uwzględnia żadnych zasad i świadczy o niskiej kulturze językowej autora, człowieka używającego takich form wyrazów. Oczywiście, w rozwoju społeczeństw zdarzają się okresy, kiedy kultura jako całość, a także kultura poszczególnych jednostek nie jest najwyższego lotu. I jeżeli ta forma zapożyczeń znajdzie uznanie u niektórych autorów i czytelników, to nie ma na to rady.

Należy podkreślić, że takie postępowanie jest częściowo uzasadnione w językach, w których odmienny alfabet uniemożliwia utrzymanie oryginalnej pisowni zapożyczeń. W języku polskim jednak problem ten nie występuje. Dlatego, na przykład, mówimy **woda kolońska**, a nie **o-de-koloń**.

I na tę zasadę nie mogą mieć wpływu ani zalecenia ISO, ażeby używać nazw międzynarodowych (notabene, nie nazwy, ale pojęcia powinny być umiędzynarodowione), ani reguła dopuszczania synonimów, o których pisze red. Klepacz. Są to reguły godne przestrzegania i trudno z nimi się nie zgodzić, ale żadna zasada ani norma nie może mnie zmuszać, abym pisał **tajmer**, **teletajp**, **displej** i **interfejs** zamiast **timer**, **teletype**, **display** i **interface** (a także digitajzer zamiast digitizer).

Przy zapożyczaniu wyrazów istnieje jednak jeszcze inne niebezpieczeństwo, na które warto zwrócić uwagę, bo jest ono poważniejsze i groźniejsze w skutkach. Jeżeli ktoś używa wyrazów **teletajp** lub **interfejs**, można to łatwo rozpoznać i zakwestionować jako błąd językowy. Często jednak wyrazów obcego pochodzenia, które są akceptowane, używa się niezgodnie ze znaczeniem. Nie znając dokładnie znaczenia obcego wyrazu, ktoś może użyć go w niewłaściwym kontekście. W ten sposób powstaje n o w o m o w a — zdania które nic nie znaczą, są puste. Tak jak tytuł tego artykułu parafrazujący znany termin informatyczny.

JANUSZ ZALEWSKI

KONFERENCJE

Komputery w medycynie

W dniach 15—17 listopada 1985 odbędzie się we Wrocławiu międzynarodowa konferencja „KOMPUTERY W MEDYCYNIE". Organizatorami konferencji są: Zrzeszenie Polskich Towarzystw Medycznych, Komisja Zastosowań Informatyki w Medycynie PAN, polska część Rady ds. Zastosowań Techniki Obliczeniowej MKETO oraz Zrzeszenie Producentów Środków Informatyki, Automatyki i Aparatury Pomiarowej MERA. Program konferencji przewiduje obrady plenarne, sesje tematyczne różnych typów oraz wystawę.

Organizatorzy zapraszają do nadsyłania w terminie do 30 kwietnia 1985 referatów o objętości do 4 stron maszynopisu. Tematyka referatów dotyczyć może wszystkich dziedzin i aspektów zastosowań komputerów w medycynie szpitalne i laboratoryjne systemy informacyjne, systemy zarządzania bazą danych medycznych, wspomaganie zarządzania służbą zdrowia, tomografia komputerowa i przetwarzanie obrazów, intensywna terapia, komputerowe wspomaganie diagnozy, terapii i badań klinicznych, informatyzacja epidemiologii, systemy ekspertowe, klasyfikacja diagnoz, informatyka w wyższej szkole medycznej i inne).

Bezpośrednią organizację konferencji powierzono ZETO WROCŁAW (50-069 Wrocław, ul. Ofiar Oświęcimskich 7/13, tel. 44-57-54 lub 44-54-31/7 w. 160 i 276, teleks 0712534 PL lub 0712533 PL), gdzie można uzyskać szczegółowe informacje oraz nadsyłać zgłoszenia referatów lub uczestnictwa.

■ Jeden z urzędników EWG w Brukseli poinformował w kwietniu bieżącego roku, że Komisja EWG zamierza nakazać koncernowi IBM ujawnienie pewnych szczegółów technicznych produkowanych przez nich komputerów, by umożliwić użytkownikom maszyn cyfrowych podłączanie ich do urządzeń produkowanych przez inne firmy. W ocenie Komisji EWG — amerykański koncern nadużywa swej pozycji monopolisty na rynku zachodnioeuropejskim, łamiąc przepisy EWG. IBM zataja szczegóły techniczne systemów, zmuszając użytkowników komupterów typu 370, 303X, 308X i 4300 do kupowania urządzeń towarzyszących wyłącznie produkcji IBM, ponieważ nie znają oni sposobu podłączenia urządzeń wytwarzanych przez firmy konkurencyjne. Opanowawszy w 50 proc. zachodnioeuropejski rynek komputerów, amerykański koncern zapewnia sobie w ten sposób monopol także w innych dziedzinach. Agencje prasowe przypominają, że próby poskromienia amerykańskiego monopolisty podejmuje się już od czterech lat. Protesty firm konkurujących z IBM nie przyniosły dotychczas rezultatu. W czerwcu Komisja EWG ma zamiar zwrócić się do krajów członkowskich ze szczegółowymi propozycjami w tej sprawie. Komisja negatywnie oceniła propozycje ustępstw zgłoszone przez kierownictwo IBM. (KI)

*

■ Metody tradycyjne nie wystarczają do opracowywania poprawnych prognoz pogody. Stan pogody przewidywany jest przede wszystkim na podstawie obserwacji chmur, które są bardzo złożonym i ciągle zmieniającym się systemem. Dane z sond meteorologicznych, próbki pobierane przez samoloty oraz dane z obserwacji dokonywanych przez radary, notujące wyłącznie ruchy chmur — stanowiące podstawę tradycyjnego przewidywania pogody — nie są precyzyjne. Korzysta się więc z metod symulacji komputerowej, pozwalających analizować działanie złożonych oraz słabo zidentyfikowanych w formie analitycznej systemów. Modele symulacyjne ruchów chmur są wykorzystywane do prognozowania pogody w kilku krajach Europy Zachodniej oraz w Stanach Zjednoczonych. Model opracowany przez naukowców z Institute of Atmospheric Sciences w South Dakota School of Mines & Technology in Rapid City jest jednym z pierwszych i — jak wykazuje praktyka — najprecyzyjniejszych modeli symulacyjnych chmur. Już w 1977 roku badania symulacyjne, z wykorzystaniem tego modelu, pozwalały przewidywać deszcze oraz burze gradowe, łącznie z określeniem wielkości kulek gradu. Obecnie możliwości modelu zostały znacznie rozszerzone. Pozwala on badać takie konsekwencje zachmurzenia (lub braku chmur), jak: związek między występowaniem upałów a wegetacją roślin, relacje między powstawaniem chmur a deszczem lub burzami gradowymi, przemieszczanie i koncentrację dwutlenku siarki w atmosferze oraz powstawanie potencjału elektrycznego chmur i wyładowania atmosferyczne. W modelu uwzględniono nawet wpływ wystrzelonych rakiet na dynamikę ruchu chmur. Wszystkie te czynniki pozwalają przewidywać możliwość wystąpienia opadów deszczu, gradu lub śniegu. Model symulacyjny składa się ze zbioru punktów odpowiadających wybranym, oddalonym od siebie o 200 m, punktom atmosfery — licząc od powierzchni ziemi aż do wysokości 20 km. Punktów tych, w jednej symulowanej sekcji, np. chmurze, jest aż 10 tys., przy czym każdy z nich opisany jest przez 15—40 wyrażeń, modelujących takie zjawiska, jak np. cień chmur na powierzchni gruntu, prędkość wiatru, wielkość chmur. Co pięć sekund symulowanego czasu rozwiązywane są równania dla wszystkich 10 tys. punktów oraz analizowany jest wpływ wyników obliczeń na zmiany warunków początkowych dla obliczeń kroku następnego oraz notowane są widoczne zmiany meteorologiczne. Podczas symulacji dwóch godzin „życia" chmury komputer powinien wykonać około 100 bilionów obliczeń. Do tego celu używany jest komputer CRAY-1, zainstalowany w National Center for Atmospheric Research, w mieście Bouldler, w stanie Kolorado. (TW)

*

■ W siedzibie firmy DIGITAL RESEARCH znajduje się komputer, który możliwościami i wyglądem przypomina LISĘ — skonstruowany rok temu stołowy komputer firmy APPLE. Nie jest to jednak LISA, ale eksperymentalny komputer działający pod nadzorem zmodyfikowanego systemu operacyjnego CP/M, napisanego w języku C i przenośnego na każdy mikroprocesor, który jest wyposażony w ten język wysokiego poziomu. Z takim systemem operacyjnym integrator systemów (ang. systems integrator) może łatwo konstruować nową LISĘ, opartą na innych procesorach 32-bitowych. Oprogramowanie to prawdopodobnie będzie dostępne w tym roku, ale już obecnie pojawiają się na rynku inne systemy, także napisane w języku C i przenośne na inne systemy operacyjne i sprzętowe, co stanowi część planu DRI mającego na celu przywrócenie tej firmie czołowego miejsca w dziedzinie oprogramowania mikrokomputerów. (MA)

*

■ W dniach 16—19 maja br. w Zamku Książ k. Wałbrzycha odbyła się III Międzynarodowa Konferencja Naukowo-Techniczna RELCOMEX'84 poświęcona zagadnieniom niezawodności i eksploatacji systemów komputerowych. Organizatorami RELCOMEX'84 były: Stowarzyszenie Elektryków Polskich, Zakłady Elektroniczne ELWRO, Politechnika Wrocławska i Uniwersytet Wrocławski. W konferencji wzięło udział ponad 200 osób z siedmiu krajów (Bułgaria, Kanada, NRD, USA, Węgry, ZSRR). Tematyka konferencji koncentrowała się wokół następujących problemów: matematyczne modele niezawodności, niezawodność i eksploatacja systemów cyfrowych, niezawodność i eksploatacja systemów mikrokomputerowych i mikroprocesorów, testowanie i diagnostyka układów logicznych i systemów cyfrowych oraz niezawodność i technologia przetwarzania w systemach komputerowych. Referat wprowadzający „O niektórych twierdzeniach ogólnej teorii niezawodności" wygłosił prof. Antoni Kiliński, nestor polskiej techniki komputerowej i prekursor badań w zakresie teorii niezawodności w Polsce. Drugim referatem sesji plenarnej było wystąpienie dyrektora naczelnego Z. E. ELWRO, mgr in. A. Musielaka, który przedstawił kierunki i rozwój produkcji Zakładów. W obradach i dyskusjach zwrócono szczególnie uwagę na to, że jak powiedział w swoim referacie wprowadzającym prof. Kiliński — „...budując matematyczne teorie niezawodności jesteśmy narażeni na przekroczenie progu konkretności i sformułowania nieadekwatnych teorii, zbyt daleko odbiegających od rzeczywistości, stanowiących abstrakcje puste, nie zawierające implikacji, jak je zweryfikować" oraz, że „... w ostatecznej instancji o niezawodności decyduje jednak nie producent, lecz infrastruktura ekonomiczna i społeczna, która powinna umożliwiać producentowi realizację postulatów niezawodnościowych". (SB)

*

■ Efektywność programowania w BASICU ułatwia pakiet o nazwie BetterBasic, który wspomaga programowanie strukturalne i ułatwia pisanie programu. Pozwala on użytkownikom tworzyć kolejne poziomy podprogramów i łączyć je w moduły wielokrotnego stosowania oraz zapewnia elastyczność nieznaną w innych wersjach tego języka. Pakiet działa około pięć razy szybciej od popularnego MicroSoftBasic, który był podstawą jego zaprojektowania. Pierwsza partia pakietu miała być dostarczona w czerwcu br. z przeznaczeniem dla IBM PC i jego odpowiedników. (MA)

*

■ Zakłady produkcji sprzętu komputerowego VEB MIKROELEKTRONIK ERFURT (NRD) rozpoczęły dostawy komputera do celów dydaktycznych. Jest to już drugie wyspecjalizowane urządzenie mikroinformatyczne produkcji tego zakładu, po wprowadzonym już na rynek mikrokomputerze do gry w szachy. Nowy mikrokomputer został zaprojektowany głównie pod kątem potrzeb dydaktycznych szkół średnich i wyższych, a służyć ma w pierwszym rzędzie do nauki języka programowania systemu mikroprocesorowego U 880. Mikrokomputer dydaktyczny umożliwia tworzenie programów dla prostych urządzeń sterujących i pomiarowych oraz samodzielnego projektowania gier dydaktycznych. Podczas pierwszej prezentacji zastosowań mikrokomputera zademonstrowano przykłady jego działania w roli woltomierza i zegara cyfrowego, urządzenia do pomiaru czasów reakcji oraz generatora melodii. Mikrokomputer został już zainstalowany w erfurckim Klubie Młodego Technika, gdzie jego członkowie będą uczyć się m. inn. techniki zapisu programów na standardowych kasetach magnetycznych. Producent zapowiedział również rozpoczęcie w br. produkcji komputerów domowych. Powyższe fakty świadczą o tym, że przemysł sprzętu komputerowego NRD bardzo szybko reaguje na potrzeby rynku, starając się kształtować swoją produkcję zgodnie z aktualnymi światowymi trendami rozwoju informatyki. (K)